2

)STGRADUATE SCHOOL
Monterey, California

THESIS

MINEFIELD SEARCH AND OBJECT RECOGNITION FOR AUTONOMOUS UNDERWATER VEHICLES

by

Mark A. Compton

March 1992

Thesis Advisor: Dr. Man-Tak Shing

Approved for public release; distribution is unlimited.

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED	1b. RESTRICTIVE MARKINGS
2a SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution is unlimited
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S)	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION Computer Science Dept. Naval Postgraduate School; 6b. OFFICE SYMBOL (if applicable) CS	7a. NAME OF MONITORING ORGANIZATION Naval Postgraduate School
6c. ADDRESS (City, State, and ZIP Code) Monterey, CA 93943-5000	7b. ADDRESS (City, State, and ZIP Code) Monterey, CA 93943-5000
8a. NAME OF FUNDING/SPONSORING ORGANIZATION; 8b. OFFICE SYMBOL (if applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c. ADDRESS (City, State, and ZIP Code)	10. SOURCE OF FUNDING NUMBERS: PROGRAM ELEMENT NO. / PROJECT NO. / TASK NO. / WORK UNIT ACCESSION NO.

11. TITLE (Include Security Classification)
MINEFIELD SEARCH AND OBJECT RECOGNITION FOR AUTONOMOUS UNDERWATER VEHICLES

12. PERSONAL AUTHOR(S)
Compton, Mark A.

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15. PAGE COUNT
Master's Thesis	FROM 04/90 TO 03/92	1992, March, 26	257

16. SUPPLEMENTARY NOTATION The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the United States Government.

17. COSATI CODES		
FIELD	GROUP	SUB-GROUP

18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
Autonomous Underwater Vehicle (AUV), minefield search, search, mine warfare, underwater object recognition, sonar classification, expert system

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

Autonomous Underwater Vehicles (AUV) are an outstanding minefield search platform. Because of their stealthy nature, AUVs can be deployed in a potential minefield without the enemy's knowledge. They also minimize dangerous exposure to manned and more expensive naval assets. This thesis explores two important and related aspects of AUV minefield search: exhaustive sensor coverage of a minefield through effective path planning and underwater object recognition using the vehicle's sensors.

The minefield search algorithm does not require a priori knowledge of the world except for user-defined boundaries. It is a three-dimensional, prioritized sub-area graph search using a ladder based methodology and an A* optimal path planning algorithm. The minefield search algorithm effectively ignores areas which are blocked by obstacles, performs terrain following and avoids local minima problems encountered by other area search solutions. The algorithm is shown to be effective using a variety of graphical simulators.

The object recognition algorithm provides autonomous classification of underwater objects. It uses geometric reasoning and line fitting of raw sonar data to form geometric primitives. These primitives are analyzed by a CLIPS language expert system using heuristic based rules. The resulting classifications may be used for higher level mission planning modules for effectively conducting the minefield search. Actual NPS AUV swimming pool test runs and graphic simulations are used to demonstrate this algorithm which was built in cooperation with Lieutenant Commander Donald P. Brutzman,

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT [✓ UNCLASSIFIED/UNLIMITED] ☐ SAME AS RPT. ☐ DTIC USERS	21. ABSTRACT SECURITY CLASSIFICATION UNCLASSIFIED
22a. NAME OF RESPONSIBLE INDIVIDUAL Dr. Man-Tak Shing	22b. TELEPHONE (Include Area Code) (408) 646-2634; 22c. OFFICE SYMBOL CS/SH

DD FORM 1473, 84 MAR — 83 APR edition may be used until exhausted. All other editions are obsolete

SECURITY CLASSIFICATION OF THIS PAGE
UNCLASSIFIED

MINEFIELD SEARCH AND OBJECT RECOGNITION FOR AUTONOMOUS UNDERWATER VEHICLES

by
Mark A. Compton
Lieutenant Commander, United States Navy
B.A., Washington State University, 1980
M.B.A., San Jose State University, 1987

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
March 1992

Author: Mark A. Compton
Mark A. Compton

Approved By: [illegible]
Man-Tak Shing, Thesis Advisor

[illegible]
Yutaka Kanayama, Second Reader

Robert B. McGhee
Robert B. McGhee, Chairman,
Department of Computer Science

ABSTRACT

Autonomous Underwater Vehicles (AUV) are an outstanding minefield search platform. Because of their stealthy nature, AUVs can be deployed in a potential minefield without the enemy's knowledge. They also minimize dangerous exposure to manned and more expensive naval assets. This thesis explores two important and related aspects of AUV minefield search: exhaustive sensor coverage of a minefield through effective path planning and underwater object recognition using the vehicle's sensors.

The minefield search algorithm does not require *a priori* knowledge of the world except for user-defined boundaries. It is a three-dimensional, prioritized graph search using a ladder based methodology and an A* optimal path planning algorithm. The minefield search algorithm effectively ignores areas which are blocked by obstacles, performs terrain following and avoids local minima problems encountered by other area search solutions. The algorithm is shown to be effective using a variety of graphical simulators.

The object recognition algorithm provides autonomous classification of underwater objects. It uses geometric reasoning and line fitting of raw sonar data to form geometric primitives. These primitives are analyzed by a CLIPS language expert system using heuristic based rules. The resulting classifications may be used for higher level mission planning modules for effectively conducting the minefield search. Actual NPS AUV swimming pool test runs and graphic simulations are used to demonstrate this algorithm which was built in cooperation with Lieutenant Commander Donald P. Brutzman, USN.

TABLE OF CONTENTS

ACKNOWLEDGEMENTS

This thesis could not have been successfully completed without the unselfish assistance of the following professionals.

MAJ Stefan A. H. Westman, Swedish Army, was particularly helpful in analyzing the minefield search concept. His creativity and analytical skills helped to keep me focused on feasible solutions. I am grateful for the assistance he eagerly provided at all hours of the day.

LCDR Donald P. Brutzman, USN, coauthored two papers and one thesis chapter. His undying motivation to pursue good scientific research was highly infectious. His ingenuity and ability to conceptualize the abstract were critical in developing the sonar analysis expert system. His NPS AUV Integrated Graphic Simulator allowed important visualization of all of the code presented in this thesis.

I am forever grateful for the close friendship of both of these officers.

The stimulating interaction with all of the members of the NPS AUV project was important to the development of this thesis. I am particularly grateful to the project leaders, Dr. Anthony M. Healey and Dr. Robert B. McGhee, for their inspiring leadership.

Dr. Man-Tak Shing and Dr. Yutaka Kanayama, my thesis advisors, are two of the most intellectually stimulating people that I have encountered. I sincerely appreciate their astute guidance and patience.

I dedicate this thesis to my wife, Deborah, for her unwavering support and love.

I. INTRODUCTION

A. AUTONOMOUS UNDERWATER VEHICLES (AUV)

Throughout the history of warfare, man's offensive weapons have progressively evolved, delivering greater destruction at longer ranges with increased accuracy. The development of such weapons has been accompanied by an increase in defensive weapon sophistication designed to deny the enemy the information necessary to target his weapons, to strike down the offensive weapon prior to terminal destruction, or to prevent the enemy from gaining ground close enough to deliver his weapons.

The introduction of calvary, archery, artillery and rocketry marks major milestones in warfare. The more recent, rocketry, required sophisticated guidance and control systems capable of withstanding the immense G forces associated with launch and maneuvering. Solid state circuitry provided the physical basis for such systems during the middle part of this century. It was the silicon computer chip introduced in the 1970's, however, which provided miniaturized advanced logic circuitry small enough to fit into small rockets and missiles and sophisticated enough to perform midcourse evaluation and guidance calculations. One of the most recent products of this technology has been the cruise missile. This weapon is able to not only store prescribed, multi-legged, flight paths but also to analyze the tactical environment enroute to its target and adjust its tactical profile according to some heuristic logic. The technology necessary for cruise missiles has found other uses in weaponry. One particular beneficiary is the autonomous underwater vehicle or AUV.

An AUV is an unmanned, untethered free-swimmer robot with sufficient on-board intelligence to perceive uncharted and unplanned situations and take action in response. An extension of this definition would include the ability to evaluate external circumstances and adjust the conduct of an entire mission as necessary to achieve the most desirable outcome.

The nature of an AUV highlights numerous qualities which make it attractive for Navy missions:

- low active sonar profile

- extremely quiet propulsion

- operates unattended in any weather

- inexpensive when compared to manned units of similar capability

- easily reprogrammed to adjust for changing mission or enemy intelligence

- potential long endurance

- does not require extensive and expensive external support

- unmanned (eliminate hazard to human)

Given this basic definition of an AUV and a list of its qualities, it is apparent that such a vehicle would be particularly well suited to many Navy mission applications. Mine detection and minefield mapping could be completely performed by an AUV in direct support of a submarine or surface ship. AUVs could conduct reconnaissance of amphibious landing areas. They could quickly clear transit lanes for fleet deployment or amphibious assault. Dropping an AUV from an aircraft could be a fast way to initiate harbor or choke point surveillance; in this role it could act as a "bell-ringer" against hostile ships. AUVs could perform large area bottom searches for high-value objects. They could deliver underwater sensors or weapons. These and potentially many other missions could be performed by an AUV at a lower cost and at no risk to either human life or expensive manned equipment.[Ref. 1]

The many benefits of an AUV in these Navy missions make it important that extensive research be conducted in both hardware and software design and that real-world type applications be thoroughly planned for and tested. The demonstration of the feasibility of these applications is critical in not only ensuring that AUVs are ready to meet fleet requirements but also in demonstrating the value of this new technology to the fleet hierarchy.

B. MINEFIELD SEARCH MOTIVATION

Mines have become an increasingly vital weapon in waging war since their first introduction during the American Civil War. They are relatively cheap and easy to produce, safe to transport, easy to deploy, difficult to detect and effective even when they do not destroy an enemy unit. The mere threat of the presence of a minefield can deny the use of bodies of water to the enemy. Mines also lend themselves to modern microchip technology. They can be exceptionally "smart", laying unpredictably dormant until the proper combination of environmental and target related detonation criteria are met.

While sophisticated mine warfare has always been conducted by the major naval powers, mines have increasingly become a weapon of choice for third-world countries for exerting military and political influence at sea. They are the most dramatic "force multiplier" in a sea-going arsenal. A strong case in point is the use of mines during the 1990-1991 Gulf War. Even though the mines deployed by the Iraq regime against United Nations Coalition Forces were generally unsophisticated, they tied up a dramatic portion of Coalition naval assets in the mine-countermeasure mission. Additionally, the *USS Princeton* (CG-59) and the *USS Tripoli* (LPH-10), both highly significant offensive assets, were taken out of action early in the conflict by Iraqi mines. The result in loss of naval and marine assets during the conflict and the over a third of a billion dollars in repair costs emphasizes the need for serious counters to this threat.[Ref. 2][Ref. 3]

AUVs are an outstanding minefield search platform under most circumstances. Because of their "stealthy" nature, they can be deployed in a potential minefield without the enemy's knowledge. AUVs minimize dangerous exposure to manned and more expensive naval assets. They can report the presence and locations of mines to their parent unit and may even be able to autonomously neutralize many types of mines.

The benefits of employing AUVs in mine warfare alone necessitate rapid research and development in this important technology.

C. THESIS OBJECTIVES

The primary objective of this thesis is to demonstrate an efficient and effective method for searching a minefield and identifying mine-like objects using an AUV. To this end it *will address the following issues:*

- What are the NPS AUV's current capabilities?
- What research efforts are directly related to minefield search and mine identification?
- What environmental, equipment and target considerations must be addressed?
- What search technique can effectively conduct the minefield search?
- What graphic based tools can be effectively used for minefield search algorithm evaluation?
- How can the minefield search algorithm be demonstrated on the NPS AUV Integrated Graphic Simulator?
- What AI methods can be used to analyze and classify sonar detected objects *including mine-like objects?*

It is a goal of this thesis to meld real-world considerations with technical solutions so that the reader feels a close association with expected "fleet" concerns while understanding the theoretical basis of these solutions.

D. THESIS ORGANIZATION

Chapter II provides a background on AUV research in naval missions and a description of the NPS AUV II project. Chapter III addresses the general minefield search problem taking into account real world considerations. Chapter IV presents an original algorithm suited to minefield area search. Chapter V addresses graphic simulation for AUV minefield search evaluation. Chapter V also presents an AUV guidance model for use in graphic simulators. Chapter VI presents an expert system which processes sonar information and postulates the identification of underwater contacts. This expert system was developed jointly with Lieutenant Commander Donald P. Brutzman, USN, of the Naval Postgraduate School. Conclusions are presented in Chapter VII.

Appendix A and Appendix B contain the two-dimensional and three-dimensional minefield search source code. Appendix C and Appendix D present the source code for the Two-Dimensional and the Three-Dimensional Graph Search Evaluation Tools. Appendix E contains AUV simulator guidance module source code. All of the above code is written in ANSI C. Finally, Appendix F presents the autonomous sonar classification expert system source code which was coauthored with LCDR Donald P. Brutzman USN.

II. BACKGROUND

A. THE NPS AUV PROJECT

The NPS AUV project began in 1987 under the sponsorship of the Naval Surface Weapons System (NSWC) at White Oak, Maryland. This research is the direct responsibility of the Mechanical Engineering, Computer Science, and Electrical and Computer Engineering departments. The first NPS vehicle, NPS AUV I, was built by Glenn Brunner of the Mechanical Engineering department. This vehicle was small, 30 by 7 by 3.5 inches, and was designed after the Navy's Swimmer Delivery Vehicle (SDV). It was a tethered vehicle which received control inputs from an IBM-AT. Model-based maneuvering control tests, including automatic identification of significant hydrodynamic characteristics were conducted in a 4 foot by 4 foot by 40 foot test tank located in the Mechanical Engineering building. The need for evaluation of a more robust and autonomous vehicle led to the design and development of the NPS AUV II.[Ref. 4] It was launched by the Superintendent, RADM Ralph W. West, Jr. at the NPS swimming pool on June 15, 1990.

1. NPS AUV II Vehicle Characteristics

The shell is an aluminum box measuring 16" by 10" by 72". The nose section is a 20" fiberglass dome housing the vehicle's sonar transducers. It is neutrally buoyant and displaces about 390 pounds. Longitudinal thrust is provided by two stern propellers driven by counter-rotating 24 volt/one-eighth horse power motors. Four cross-body tubes have been provided to house two unique thrusters currently under development. These thrusters will allow the vehicle to control vehicle posture and maintain station in the presence of underwater currents. The thrusters will also permit vertical motion, allowing the vehicle to maneuver in very limited space. Lead-acid gel batteries give the vehicle an endurance of two to three hours. Maximum speed is about two knots. There are four vertical and four horizontal independently controlled planes. The AUV's turning diameter is about 20 feet, designed to be ideal for maneuvering in the NPS large-size swimming pool. The vehicle is currently limited to approximately 10^{o} elevation angle. This limitation is center of gravity dependent. The center of gravity of the vehicle may be adjusted to meet almost any elevation requirements. Navigation sensing and control is provided by a flux gate compass, directional gyroscope, vertical gyroscope, a three axis rate gyroscope system with translational accelerometers and a paddlewheel speed sensor. Refer to Figure 1 for the NPS AUV II general schematic.[Ref. 5]

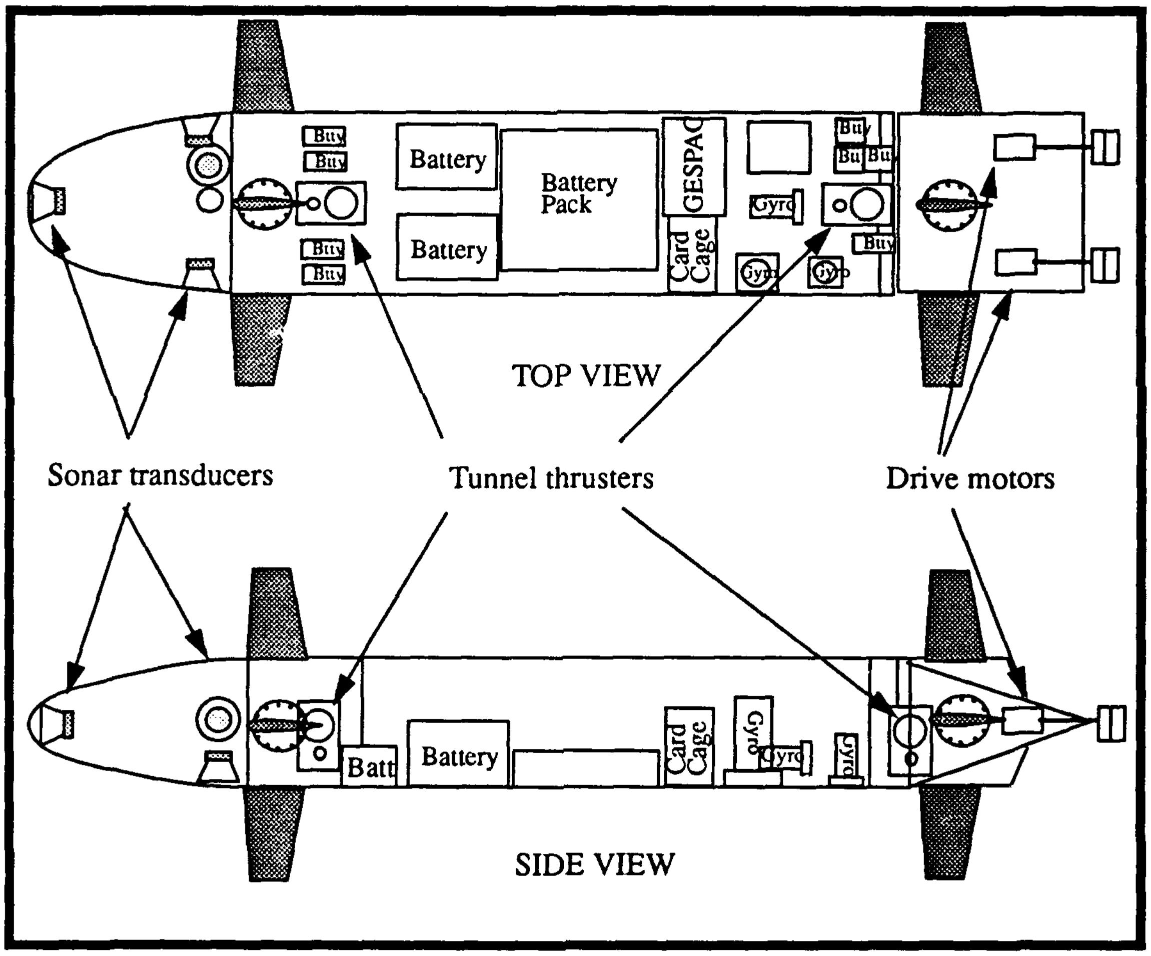


Figure 1: NPS AUVII General Schematic

Internal computer hardware consists of a GESPAC MPU 20 HF processor with a Motorola 68020 CPU, a 68881 math coprocessor, and a 2.5 Mb RAM card running at 16

MHz. Input/output between the CPU and analog control systems is through two GESDAC-2B 8-channel 12 bit digital-to-analog/analog-to-digital converter cards and a GESPIA-3A parallel interface board. Programs are loaded from a GRIDCASE 386 laptop computer via a serial connection to a 2400 bps modem. Post-mission data is downloaded through the same system.[Ref. 6]

Four Datasonics PSA-900 Programmable Sonar Altimeters are orthogonally fixed in the nose of the NPS AUV and are oriented forward, down, left and right. These transducers are fixed frequency and ultrasonic at approximately 200 KHz. The sonar range gate is selectable at 30 m or 300 m, and pulse length is 350 µs. Normal pulse repetition rate is 10Hz. Sonar beamwidth is seven degrees and range resolution is 1 cm at 30 m.[Ref. 7] Because of its high frequency resolution, this sonar is well suited as an obstacle avoidance sonar. It is not a particularly good object detection sonar because of its narrow beamwidth.

2. NPS AUV II Software Characteristics

a. Mission Execution Systems

The NPS AUV II mission execution software design and functions as presented in [Ref. 5] are shown in Figure 2.

(1) Mission Planner/Replanner System Module. This system was originally designed by S. Kwak and S. Ong using a KEE expert system shell [Ref. 8]. It is currently run off-line on a stand alone Symbolics 3675 Lisp Machine. The Mission Planner takes mission inputs from the human operator. It then decides which planning algorithm to execute depending on its internal knowledge base. In a minefield search mission, the Mission Planner would choose an algorithm similar to the minefield search algorithm presented in this thesis. Real-time updates from the Environmental Model Data Base, Obstacle Avoidance Decision Maker and Vehicle Condition Monitoring Sensors would modify the conduct of the mission execution as dictated by the Mission Replanner. The minefield search algorithm proposed in this thesis would reside inside the Execute Mission module.

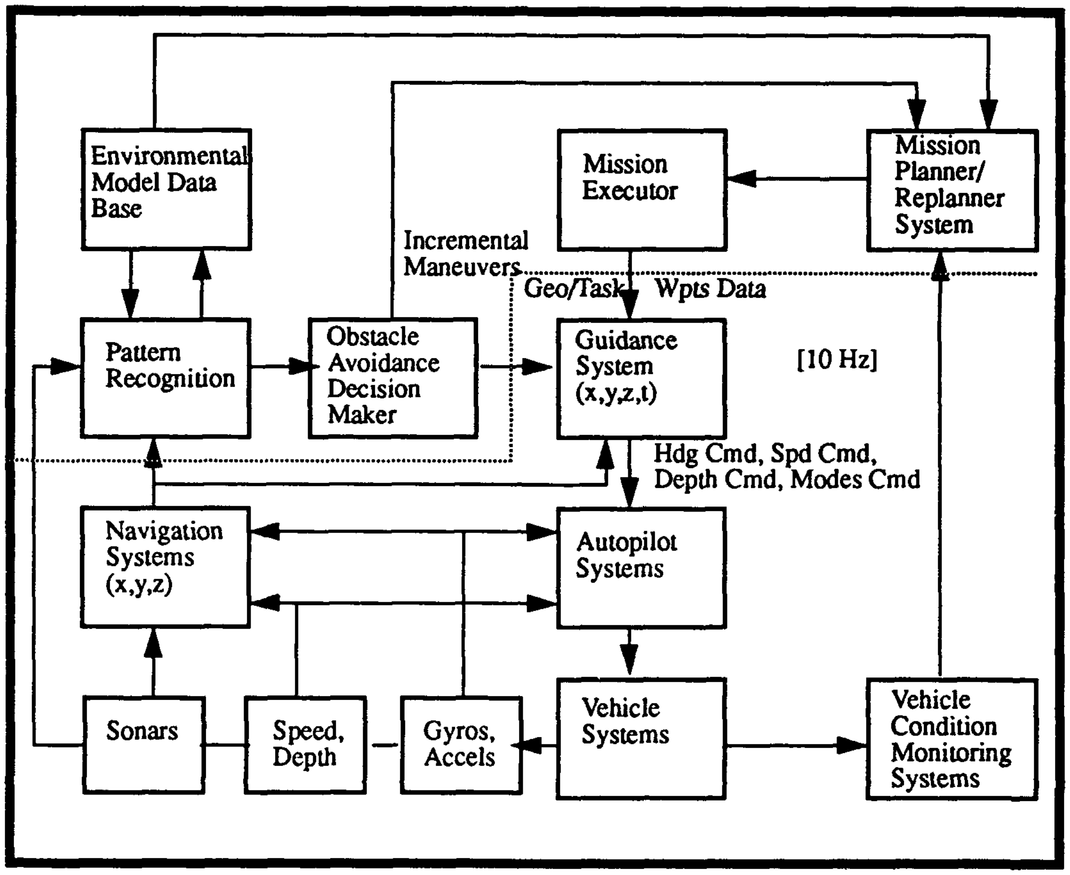

Environmental Model Data Base
Mission Executor
Mission Planner/ Replanner System
Incremental Maneuvers
Geo/Task
Wpts Data
Pattern Recognition
Obstacle Avoidance Decision Maker
Guidance System (x,y,z,t)
[10 Hz]
Hdg Cmd, Spd Cmd, Depth Cmd, Modes Cmd
Navigation Systems (x,y,z)
Autopilot Systems
Sonars
Speed, Depth
Gyros, Accels
Vehicle Systems
Vehicle Condition Monitoring Systems

Figure 2: NPS AUVII Mission Execution Diagram

(2) Mission Executor Module. Under most circumstances, a series of waypoints which make up the mission execution plan $[W_1,..., W_n]$ would be sent from the Mission Planner/Replanner to the Mission Executor. The Mission Executor would feed these waypoints to the Guidance System as required to carry out the mission. In the proposed minefield search algorithm the Mission Executor serves to relay only the current and the next waypoint positions (W_i,W_{i+1}) to the Guidance System. Thus this module would be essentially unused.

(3) Guidance System Module. This module combines commands for the path and position to be followed or tracked $[x(t), y(t), z(t), t]$, and other attitudinal

requirements with navigational estimates of true position $[\hat{x}(t), \hat{y}(t), \hat{z}(t)]$ and orientation to generate heading, speed and depth commands to the autopilot. A kinematic Euler method algorithm for computing vehicle changes in position and posture $([x, y, z, \phi, \theta, \psi])$[1] is presented in Chapter V. This model serves as the "black box" guidance system for graphic simulation mission playback purposes.

(4) Pattern Recognition Module. This module receives raw sonar data, processes the data to classify objects and sends object classification data to the Environmental Model Data Base as well as to the Obstacle Avoidance Decision Maker. The sonar analysis expert system described in Chapter VI resides in this module.

(5) Other Modules. The Autopilot System is responsible for controlling the vehicle's dynamics to carry out commands sent from the Guidance System. Vehicle (and ancillary) Systems provide a variety of vehicle related inputs to Navigation, Autopilot and Mission Replanner Systems. Positional estimates are provided by the Navigation System module as (x, y, z) coordinates. The Obstacle Avoidance Decision Maker module analyzes information from the Pattern Recognition module to determine if vehicle guidance action is needed to avoid obstacles or to react in other ways as determined by the Mission Planner/ Replanner. Finally, the Environmental Model Data Base stores *a priori* environmental data as well as data obtained during the conduct of the mission.

b. Dataflow Diagram

The NPS AUV II data flow diagram as describe by C. A. Floyd [Ref. 6] is provided in Figure 3. This diagram provides further enhancement of the Mission Execution Systems diagram described above and along with the data dictionary is the blueprint for AUV software design.

1. Note that ϕ corresponds to vehicle roll, θ to elevation, and ψ to azimuth.

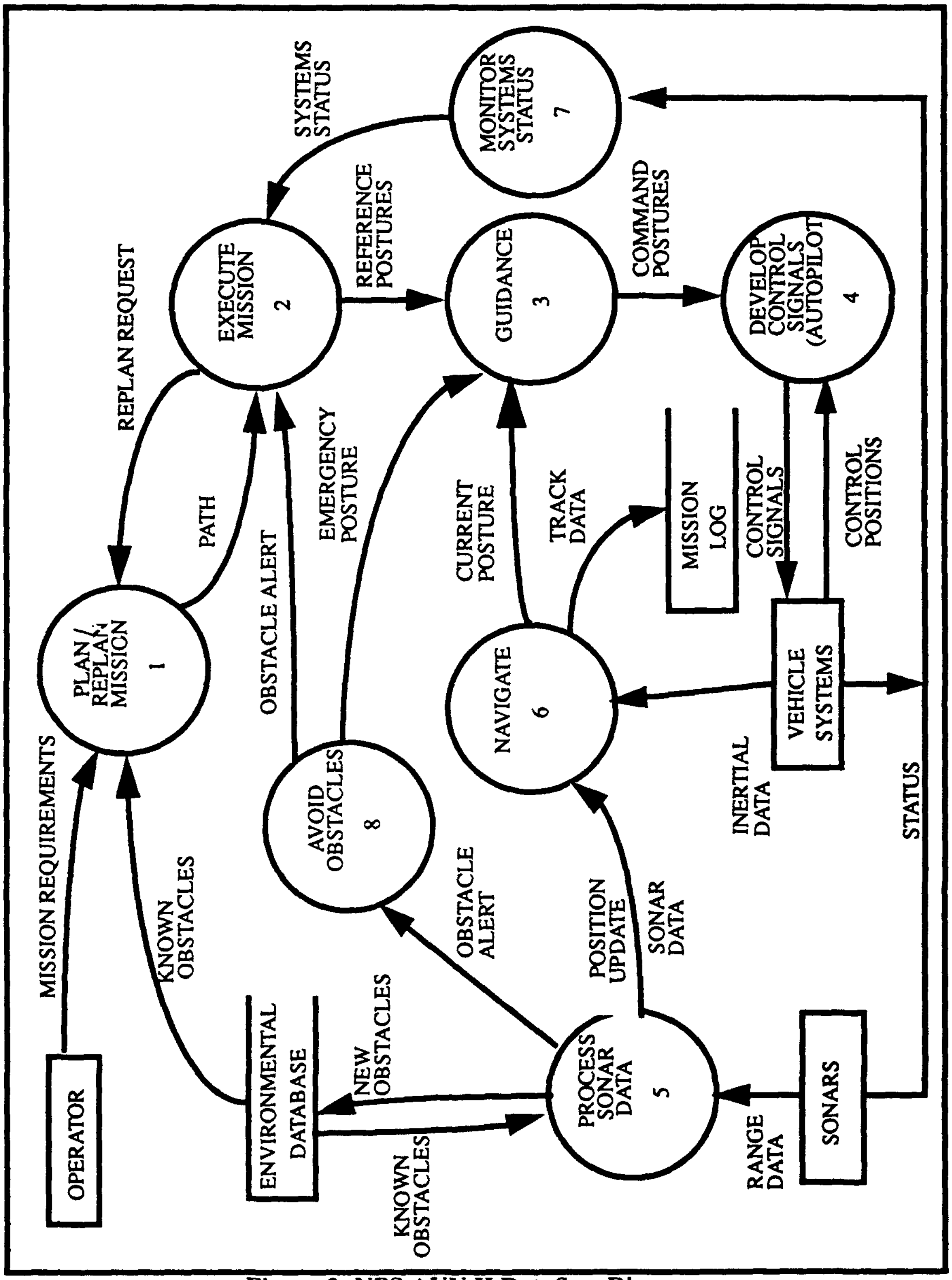


Figure 3: NPS AUV II Dataflow Diagram

3. NPS AUV II Graphic Simulators

The original NPS AUV non-graphic simulator was designed by R. J. Boncal with 3D modifications using dynamics from the U.S. Navy's Swimmer's Delivery Vehicle (SDV) by D. L. MacPherson [Ref. 9][Ref. 10]. NPS AUV-SIM1 was the result of a graduate graphics project by D. Marco, R. Rogers, and M. Schwartz. It used revised SDV equations of motion as modified by R. J. Boncal to model NPS AUV I [Ref. 9]. The NPS AUV-SIM2 modified the original vehicle graphics to reflect the geometry of the current vehicle, NPS AUV II. A major revision by T. A. Jurewicz encapsulated the AUV as a rigid body using object oriented programming techniques and modified the equations of motion and hydrodynamics to reflect the actual AUV vice the modified SDV [Ref. 11]. All of these simulators responded to control inputs from the mouse pad as opposed to using waypoint following techniques or mission file playback.

The simulator modifications by C. A. Floyd allowed for post-mission replay of actual AUV pool-side missions, waypoint following using cubic spiral guidance and obstacle avoidance and terrain following using sonar data [Ref. 6][Ref. 12].

A spatial tracking guidance scheme developed by Y. Kanayama was implemented in the NPS AUV II simulator by C. Magrino [Ref. 13]. Spatial tracking provides for a smooth return to a path defined by two waypoints. This algorithm is especially useful for maintaining acoustic search path continuity.

The latest graphic simulator, built by D. P. Brutzman, is currently under development. It is an integrated simulator in that it allows AUV mission execution code to be run on a twin AUV computer system which is maintained in the laboratory. This integrated graphic simulator will allow pre-mission evaluation of code and graphic simulator playback prior to the vehicle entering the water. The simulator does not currently have an imbedded guidance system for computing vehicle motion. Rather, it relies on a file of vehicle posture inputs which have been computed by some external module. Therefore, one aspect of this thesis addressed in Chapter V is the development of a "black box" which can take waypoint data generated by the minefield search algorithm, produce a continuous vehicle path between these waypoints, and send the resulting vehicle postures to the integrated graphic simulator for display. [Ref. 14]

B. RELATED AUV RESEARCH

There is currently a tremendous amount of AUV research being conducted throughout the world. Although each of these projects has aspects which are pertinent to the NPS AUV program, the research being done at the Defense Advanced Research Projects Agency (DARPA) represents the vanguard in minefield search technology. Aspects of this program dealing with mission pre-planning, on-board planning decision and optimization, and scene recognition are directly addressed in this thesis. The other studies discussed in this section center on search algorithms which may be used in the conduct of a minefield search.

In 1986, DARPA was tasked by the President's Blue Ribbon Commission on Defense Management to investigate prototype programs not emphasized by the services. As part of this investigation, DARPA analyzed each of the U.S. Navy's operational capabilities. It was determined that there existed an important need to research the feasibility of using Unmanned Underwater Vehicles (UUVs) in naval missions. The resulting UUV Master Plan identified the following missions for further research [Ref. 15]:

- Tactical Acoustic System (TAS)
- Mine Search System (MSS)
- Remote Surveillance System (RSS)

1. Test-Bed UUV

In support of this research, DARPA in conjunction with Charles Stark Draper Laboratories (CSDL) of Cambridge, Massachusetts has developed a working test-bed UUV. This vehicle is particularly interesting because of its maximum depth and maximum speed and endurance. The maximum depth of 1000 to 1500 feet would allow the vehicle to operate below the sonic layer depth, generally increasing sonar ranges for bottom or deep water sensor coverage. The vehicle's endurance is 24 hours at 10 knots (maximum speed). To provide some perspective, this would allow 240 nautical miles of vehicle transit, enough to cover nearly 600 square nautical miles of ocean in an expanding square type search pattern with 2 nautical mile track spacing.

2. Mine Search System Program

The MSS development contract was awarded to Lockheed Missiles and Space, Co. in January 1990. This program focuses on the UUV's ability to guide submarines or surface ships through minefields in a semi-autonomous mode (fiber-optic communication link or acoustic data link) or independently search a minefield in an autonomous mode.

3. Autonomous Minehunting Technology (AMT) Program

This project at CSDL is an important part of the MSS effort. The AMT program goal is to "Develop and demonstrate the enabling technologies for autonomously detecting, localizing, and classifying sea mines (through):

- signal extraction and image recognition,
- data fusion and mission management, and
- ultra-high resolution sensing." [Ref. 16]

Critical to these subgoals is the state of sonar technology. Many important sonar projects are underway at CSDL which may dramatically affect the quality of sonars to be carried by AUVs. Of particular interest is the use of the "mechanical hydrophone." This project exploits semiconductor fabrication technology to mass produce hydrophones smaller than the head of a pin. Hundreds of these chips could be formed into an array and placed on the body of the AUV or towed on a cable. This would allow the vehicle to analyze its environment in a full spherical fashion, eliminating gaps formed by many of todays sonars.[Ref. 17]

This thesis directly addresses several aspects of the MSS and AMT programs. It proposes a sensor-interactive mission planner algorithm to augment the MSS requirements for autonomous exploration of a minefield. It directly addresses AMT requirements for signal extraction, image recognition and data fusion by proposing and developing an expert system for autonomous sonar classification.

C. RELATED SEARCH ALGORITHM CONSIDERATIONS

1. Spatial Representation

According to G. Dudek, et al., the past work on spatial representations for robotic exploration can be broadly classified into three categories, metric, probabilistic, and graph-based [Ref. 18]. It was the consideration of the applicability and past implementations of these models which led to this author's choice of graph-based representation for the minefield search algorithm.

a. Metric Representations

This category uses a metric representation of space where the features are explicitly associated with Cartesian coordinates. The purpose of the metric representation is to enable the robot to plan and execute navigation paths with minimal dependance on external sensors. While these graphs can be used for navigation, it is unclear how to represent sensor coverage in such a way as to ensure that the entire area is searched.

b. Probabilistic Representations

Probability distributions associated with spatial coordinates are used to explicitly represent and manipulate spatial uncertainty. Probabilities of the vehicle being at a certain location are based on available sensor inputs. These concepts are used to determine not only a robot's location but also the probable path of least resistance to goal locations.

c. Graph-based Representations

This method uses topological models for representing space and approaches map learning as a graph theoretic problem. The work of G. Dudek, et al., has centered on a graph-based method which eliminates the need for a topological model, assuming that by searching all accessible space the vehicle will form its own world map. Vertices represent significant navigational events while undirected edges represent the paths between these events. This work assumes that it is not only important that every graph vertex be visited but also that the way in which the vertices are connected must be determined. Markers are used to indicate which vertices have been visited. The algorithm assumes that the vehicle has no inertial guidance, nor a compass, nor any other mechanism for determining absolute orientation. Therefore it is unable to distinguish one edge leaving a vertex from another by standard navigational means. The vehicle must, therefore, also be able to determine the relative order of the edges leaving a particular vertex. Thus, edges must also be labeled. While this work is important for vehicles with limited navigational capabilities, it is unnecessarily sophisticated for vehicles like the envisioned AUV which will have a capable navigational suite.[Ref. 18]

The graph-based representation idea does allow all of the variables associated with the minefield search problem to be treated in discrete terms with the exception of actual vehicle motion which must be handled with a near-continuous algorithm presented in Chapter V.

2. Vertex Based vs. Edge Based

Whether to use a vertex or an edge based logic in the minefield search algorithm was a critical consideration in this thesis.

Vertex based search routines are normally chosen when the importance of arriving at a specific vertex is more important than maintaining a route between vertices. Hamiltonian path planning methods would generally be employed in these instances.

Generally, edge based search routines are chosen because of the importance of the vehicle following the edge as opposed to simply reaching the vertices. Edges may be directed or undirected. Generally in the open ocean environment edges will be undirected. Eulerian path planning methods are normally employed in order to ensure that the required edges are transited.[Ref. 19] These methods seem very applicable when conducting area searches with long, parallel tracks (see Figure 5). Problems arise, however, when unexpected obstacles are encountered. It is for this reason that the minefield-search algorithm described in Chapter IV is vertex based.

3. Search and Path Planning Approaches

It is significant to note that while there are a vast number of search and path planning strategies delineated in the literature, limited research has been dedicated to mapping or searching an entire area. This thesis proposes a three-tiered search strategy and path planning approach, shown in Figure 4, for conducting the minefield search. While it

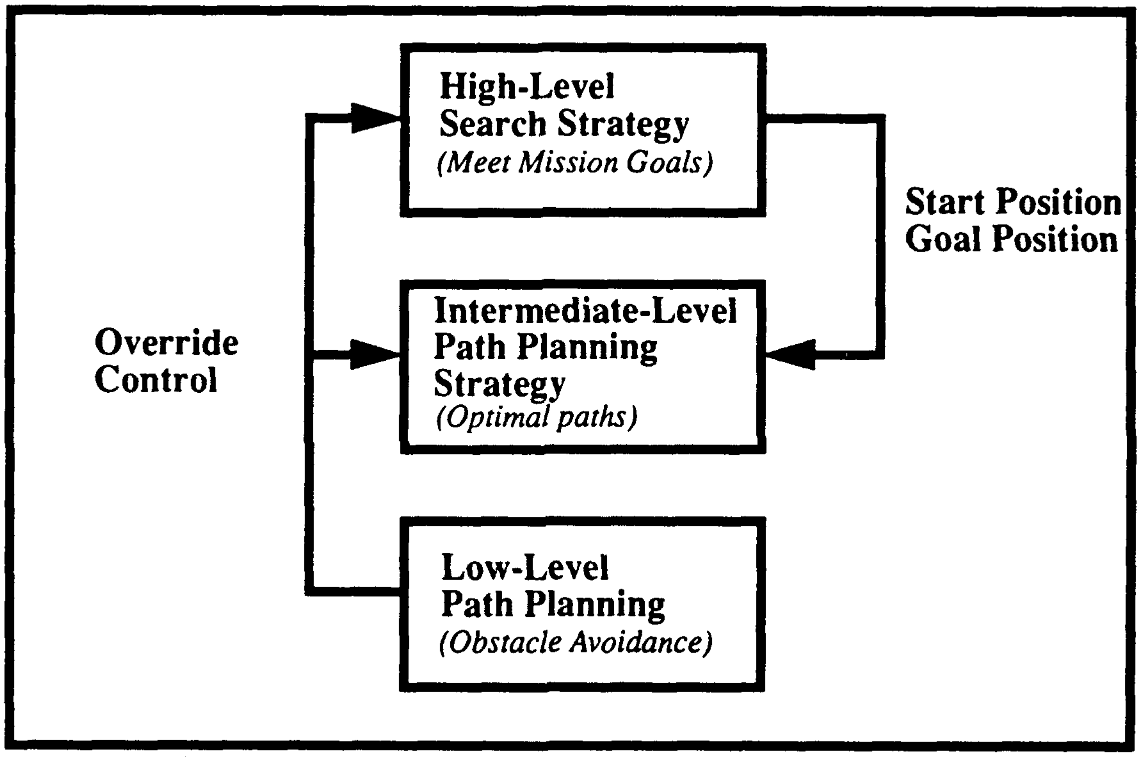


Figure 4: Three-Tiered Search and Path Planning Approach

is possible to search an entire area using a random technique, the lack of efficiency makes this method prohibitive. It is therefore desirable to use a high-level search strategy which ensures that mission goals are met in an efficient manner under most foreseen circumstances. Because of the need of the vehicle to maneuver around obstacles to reach the next goal there must also be an intermediate-level path planning strategy designed to maneuver the vehicle around these obstacles in an efficient manner. Once these maneuvers are completed the vehicle must then return to the high-level strategy. Both the high-level and intermediate-level strategies are handled in module 1 (Plan/Replan Mission) and/or module 2 (Execute Mission) in the AUV II dataflow diagram in Figure 3. A minefield search algorithm integrating these two levels is proposed in Chapter IV. The low-level path planning involves maneuvering of the vehicle in close proximity to unforeseen obstacles to avoid collision. The functions of this level are handled between modules 8 (Avoid Obstacles) and 2 (Execute Mission) in the dataflow diagram and are not addressed in detail in this thesis.

a. High-Level Search Strategies

The choice of a high-level search strategy is very mission dependent. There is a basic theme however which requires that the overall probability of detecting the target or targets of interest is maximized while minimizing the utilization costs of precious assets. Three examples in Figure 5 illustrate this point[Ref. 20].

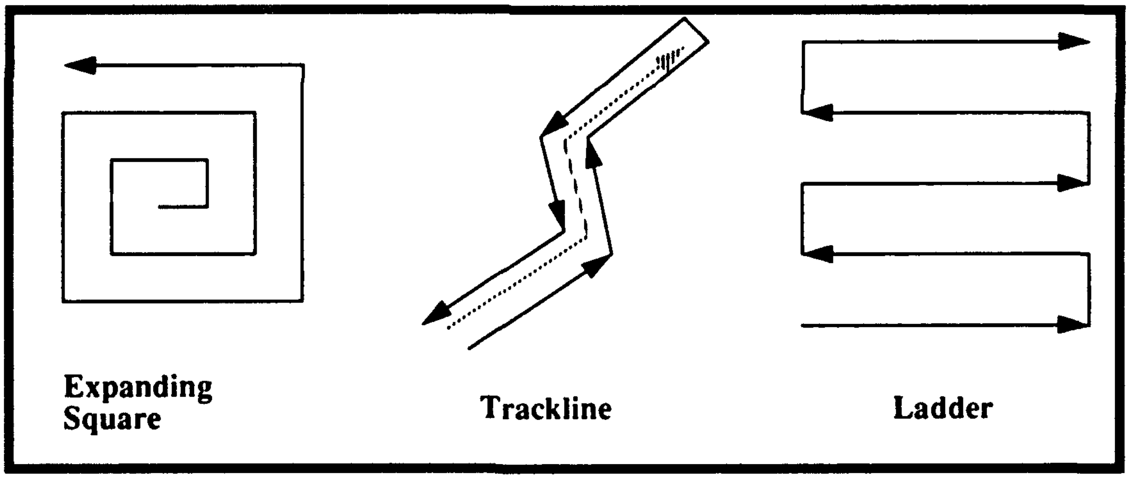


Figure 5: Search Pattern Examples

(1) Expanding Square Search. This search strategy begins at a location which is at the highest area of probability for detecting the targets of interest which generally have unknown courses. It is then expanded outward in all directions. It has the benefit of gaining contact on the targets in the earliest time possible given accurate location

intelligence. This type of search strategy may be applicable in some minefield search scenarios where it is necessary to clear mines from a datum.

(2) Trackline Search. This search strategy is only used when the intended track of the target is known. It may be employed to guide a ship or submarine through a minefield but it is generally not usable for minefield search missions.

(3) Parallel Track (Ladder) Search. This search strategy is generally employed when the search area is large, when the targets have only an approximate location accompanied by a predicted course, when uniform area coverage is desired, or any combination of the above. This search strategy and numerous variations lend themselves well to larger area minefield searches and will be addressed in detail.

Generally, high-level search strategies will be divided into a sequence of intermediate goals which normally correspond to locations where a pre-planned vehicle course change is required.

b. Intermediate-Level Path Planning Strategies

Once an intermediate goal is determined by the high-level search strategy, the vehicle must determine the best path to reach this goal. This is a straight forward task in an obstacle free environment but requires a clever algorithm to be efficient when obstacles are present. There is an abundance of search strategies which may be used at the intermediate level for planning a path from a starting point, around obstacles to the intermediate goal. The most commonly used are addressed below. [Ref. 21]

(1) Depth-First Search (DFS). In the DFS strategy, a start state is visited followed by a successor state which is chosen by a predefined heuristic. A successor state to the successor state is visited and so on until a state with no successors is reached. The search vehicle is then backed out to a previously visited state which has an unvisited successor. This pattern continues until all states have been visited. The DFS does not require an agenda of states to be visited nor any cost related functions.

Advantages:

- Easy to implement

Disadvantages:

- Not guaranteed to find goal

- Inefficiencies in backtracking lead to slow execution in hard problems

(2) Breadth-first Search (BFS). The BFS has a start state from which it visits all immediate successors according to some heuristic order. The successors of each of these states are then visited and so on until the goal is found. This search requires an agenda of successors to be visited but does not require any cost related functions.

Advantages:

- Easy to implement

- Not subject to endless loops

- Guaranteed to find goal state if reachable

- First path found to goal state is shortest in terms of layers

Disadvantages:

- Slow execution in hard problems

- Requires extra storage in the form of an agenda

- Minimizes numbers of layers traversed not the total distance

(3) A-star Search. This agenda-based search strategy uses both a cost function and an evaluation function. For a route planning problem such as the one presented in this thesis, the cost is likely to be the cumulative distance traversed from the start state to the state being evaluated. The evaluation function is likely to be the straight-line distance from the state of interest to the goal state.

Advantages:

- Guaranteed to find the lowest-cost (Optimal) path[2]

- Efficient for complex problems

Disadvantages:

- More complex to implement

- Requires longer computational time

(4) Hill-climbing Search. This is the evaluation-function variant of the DFS. It has the advantage over DFS of generally reaching the goal sooner.

(5) Best-first Search. This is the evaluation-function, agenda based version of the BFS. It picks the best-evaluation state of those anywhere in the search graph whose successors have not yet been found, not just a state at the same level as the last state. Its advantage over BFS is that it tends to reach the goal sooner under most circumstances.

While this list is not all-inclusive, it does provide the major concepts and considerations when choosing an intermediate-level search strategy.

2. E. Hart, et al. provide an extensive proof that the A-star algorithm is guaranteed to find an optimal path from start to goal given a proper evaluation function [Ref. 22].

III. MINEFIELD SEARCH PROBLEM CONSIDERATIONS

There are many missions of both offensive and defensive natures which are performed in mine warfare. Likewise, there are many corresponding mine-countermeasure missions. This section outlines the primary mine-countermeasure missions in which the AUV can be expected to participate.

The mine-countermeasure mission defines the target sought, the environment operated in and the equipment required. Each of these categories must be considered when developing minefield search problem solutions. These categories are also discussed in this section along with navigational considerations which are a combination of both environment and equipment considerations.

A. THE MISSION

It can be expected that the military will be the predominant user of an AUV mine-countermeasure weapon whether in support of military operations or "right-to-passage" civilian shipping. The following naval warfare missions define the parameters in which such an AUV would be expected to operate.

1. Battle Group Support

Most battle group missions can be expected to be carried out in deep-ocean areas where the likelihood of encountering a significant mine warfare threat is small. There are, however, a large number of cases where battle group missions are carried out in more high-threat mine warfare areas. The most recent example is in the 1990-1991 "Gulf War" where significant United Nations Coalition battle group assets conducted operations in the Red Sea, the North Arabian Sea, and the Persian Gulf. Each of these bodies of water are ideally suited for mine warfare operations. Additionally, the battle group was especially vulnerable while transiting to and from these areas through the very narrow and shallow Suez Canal, the Bab el Mandeb Straits and the Straits of Hormuz.

2. Amphibious Operations Support

Amphibious missions generally begin in moderately deep ocean staging areas where large amphibious ships launch their landing craft. These landing craft transit through shallow water areas, into the surf zone and finally ashore. Mine warfare is particularly effective in deterring these types of missions because control of both the waters leading into the beach head and the skies overhead is generally of a tentative nature due to the proximity to the enemy. It is for this reason that conventional mine-countermeasure weapons are often unable to freely search these areas. Additionally, many navigation hazards can be expected including both unplotted natural hazards as well as enemy built hazards.

3. Submarine Force Support

Like the battle group, submarines are also extremely vulnerable to mine warfare while transiting narrow and shallow bodies of water. Because the submarine depends upon remaining undetected for survival, its inability to overtly conduct mine-countermeasure operations is a severe handicap. It would be an exceptional advantage to be able to launch and recover a mine-hunting AUV from the torpedo tubes of a submerged submarine. This would allow the submarine to remain covert while surveying safe routes through potential minefields. A scene from a graphic simulation program depicting such operations is shown in Figure 6.[Ref. 23]

4. Harbor Patrol Support

AUVs may also be used for defensive harbor patrol operations. This environment is typically dense with both man-made and natural obstacles. Effective path planning and obstacle avoidance will be particularly important in this mission.

In summary, an AUV can be expected to support any of these missions. It must be able to flexibly operate under a variety of circumstances. It must be able to search moderately deep bodies of water as well as very shallow or even surf zone areas. It must be able to search narrow, restricted maneuvering areas as well as large, open bodies of water. It must search familiar as well as unfamiliar areas with the possibility of encountering

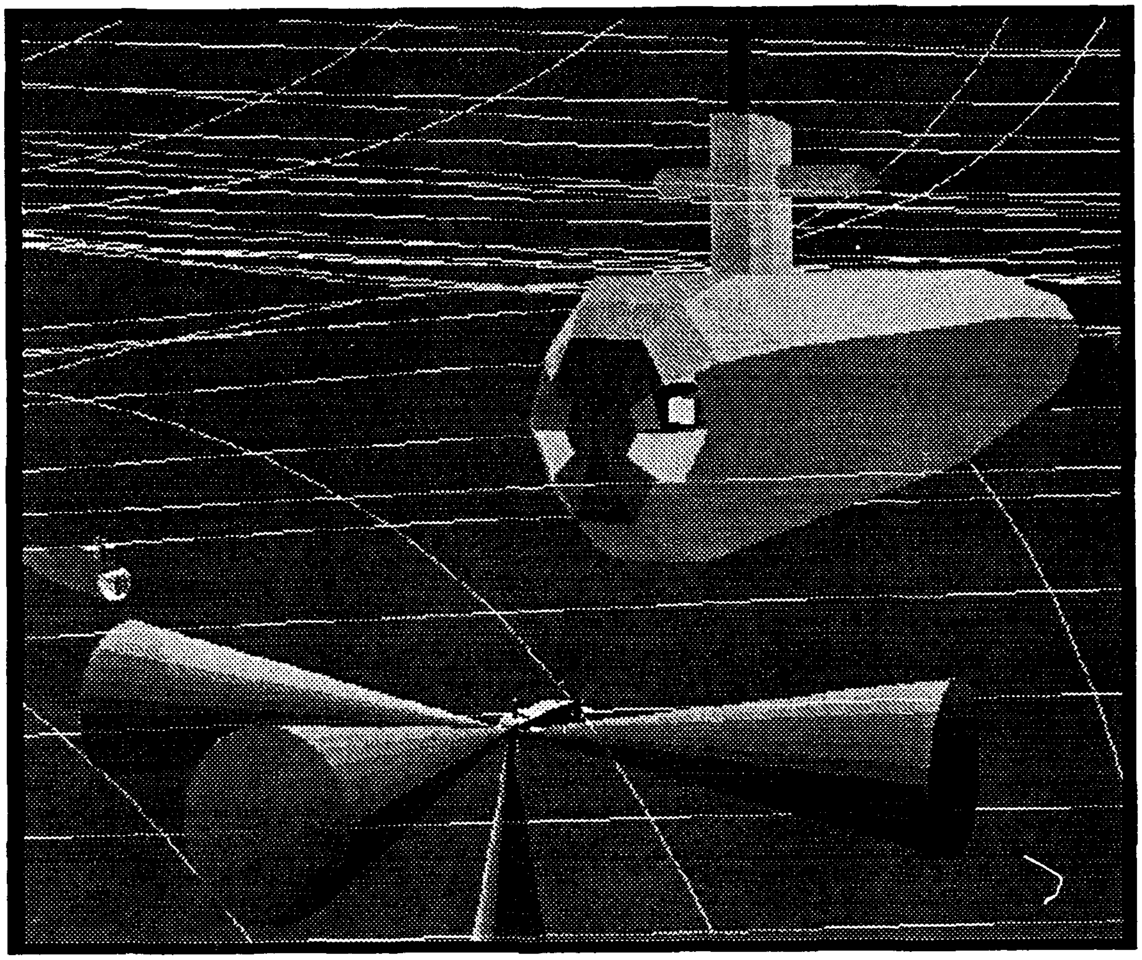


Figure 6: Graphic Simulation of Submarine Launched AUV

geometrically complex obstacles and even "blind alleys"[1] through which it must maneuver. It must also be able to return a usable "map" of what it has observed to its parent platform.

B. TARGET CHARACTERISTICS

There are two perspectives from which to view the "target of interest" for AUV mine-countermeasure missions. The first perspective views the mine as the target. This generally focuses on the sonar detection and identification of the mine. The autonomous sonar classification expert system addresses the identification aspect of this issue. The second perspective views the effective coverage and mapping of an area as the target. This

1. A blind alley is a horizontally oriented object which is open in the center.

approach assumes that determining areas as either free or hazardous is the primary goal and that specific mine identification can be handled within a different logic module. These are primary concerns in the minefield search algorithm.

There are many ways to categorize mine types: offensive or defensive; bottom, moored or floating; contact or influence; method of delivery; etc.[Ref. 2] The research in this thesis is predominantly concerned with the physical characteristics of the mine for identification considerations and the location of the mine for search considerations.

Table A highlights features of selected U.S. underwater mines [Ref. 2].[2]

TABLE A: SELECTED U.S. UNDERWATER MINES

Mine	Type	Max. Depth	Length	Diameter
Mk 52	ASW Bottom	600 ft.	5 ft. 1 in.	2 ft. 9 in.
Mk 55	ASW Bottom	600 ft.	6 ft. 7 in.	23.4 in.
Mk 56	ASW Moored	0-1200 ft.	9 ft. 5 in.	23.4 in.
Mk 57	ASW Moored	1148 ft.	10 ft. 8 in.	21 in.
Mk 36	Bottom	300 ft.	7 ft. 5 in.	15 in.
Mk 40	Bottom	300 ft.	9 ft. 9 in.	22.5 in.
Mk 41	Bottom	300 ft.	12 ft. 5 in.	25 in.
Mk 60	ASW Captor	3000 ft.	12 ft. 1 in.	21 in.
Mk 62	Bottom	300 ft.	7 ft. 5 in.	10.8 in.
Mk 63	Bottom	300 ft.	9 ft. 5 in.	14 in.
Mk 64	Bottom	300 ft.	12 ft. 8 in.	18 in.
Mk 65	Bottom	300 ft.	9 ft. 2 in.	29 in.
Mk 67	SLMM	328 ft.	13 ft. 5 in.	21 in.

2. Captor (Encapsulated Torpedo) is laid in deep water by aircraft or submarine. Upon submarine detection, launches encapsulated Mk 46 acoustic homing torpedo.
Submarine-Launched Mobile Mine (SLMM) is a self-propelled torpedo-like mine which permits covert mining by submarines. It is also a shallow-water bottom mine for use against surface ships.

These weapons typify the types of mines used by the more advanced military industrial countries. Third-world countries typically use mines of the WWI or WWII vintage but are increasingly employing more modern and sophisticated mines.Table B gives two typical mines employed in Middle East conflicts in recent years.

TABLE B: SELECTED MIDDLE EAST UNDERWATER MINES

Mine	Type	Max. Depth	Length	Diameter
M 08	Moored	110 m.	~ 1.5 m.	~ 1.5 m.
Sigeel/400	Bottom	-	~ 0.9 m.	~ 0.9 m.

Since most mines do not emit appreciable noise they are generally detected through active acoustic means. It is the difference in material density between the water and the components of the mine which cause active sonar pulses to reflect back to the sonar receiver. The size of the mine has a direct relationship to its detectability and can also be used as a numerical identification characteristic. While there are a few examples of exceptionally large mines and particularly small mines, most mines have between 10 sq. ft. and 40 sq. ft. of cross-sectional area.

C. ENVIRONMENT CONSIDERATIONS

The environment in which the minefield is placed is an important determinant of how to best employ an AUV mine-countermeasure weapon. The environment can be defined as a function of geographic characteristics and ocean acoustic properties.

1. Search Area Characteristics

The characteristics of typical minefield locations define the environment in which an AUV mine-countermeasure weapon must operate.

a. Shallow and Deep Water

Although mine fields can be placed in deeper ocean areas, they are generally placed in shallow and restricted navigational waters. Weapons of this nature are usually more effectively used in "choke point" type operations where the enemy vessel must transit through a narrow channel to reach its destination. Because choke points are normally relatively shallow and because mines are more effective when exploded in close proximity to their target, most mines are made to operate in relatively shallow water areas. Table A and Table B indicate selected mine operating depths. The approximate dimensions of selected areas where minefields could be expected to be deployed in selected times of crisis are depicted in Table C.[Ref. 25][3] In each of these bodies of water the depth is far less than

TABLE C: SELECTED GEOGRAPHIC AREAS FOR MINEFIELD OPERATIONS

Area Name	Expected Type of Minefield	Length	Least Width	Least Depth
Straits of Bab el Mandeb	ASW/ASUW	42 nm	9 nm	45 m
Suez Canal	ASUW	86 nm	290 m	19 m
Straits of Gibraltar	ASW/ASUW	33 nm	8 nm	50 m
Straits of Mal-acca	ASW/ASUW	520 nm	8 nm	26 m
Straits of Dover	ASW/ASUW	20 nm	18 nm	20 m

the range capabilities of most active sonars expected to be deployed aboard AUVs. This would allow the minefield search problem to be reduced to a two-dimensional search algorithm.

Another area where extensive ASW mine warfare operations may be conducted in the case of a major conflict with Russia is in the GIUK Gap[4] in the North Atlantic. Here, water depths average about 1800 meters and the gap is about 100 nautical miles wide to the northwest and about 450 nautical miles wide to the southeast of Iceland.

3. ASW equates to Antisubmarine Warfare and ASUW equates to Anti-Surface Unit Warfare.
4. GIUK equates to Greenland, Iceland, and the United Kingdom.

At these depths, the AUV would most likely need to conduct three-dimensional searches given sonar range limitations and deep acoustic sound channel effects. Figure 7 provides a vertical comparison of AUV sonar coverage in the Straits of Bab el Mandeb and the GIUK Gap given an 300 meter sonar range gate.[5] It is apparent that a two-dimensional search algorithm will handle most minefield requirements but that a three-dimensional search algorithm would add deep water search flexibility.

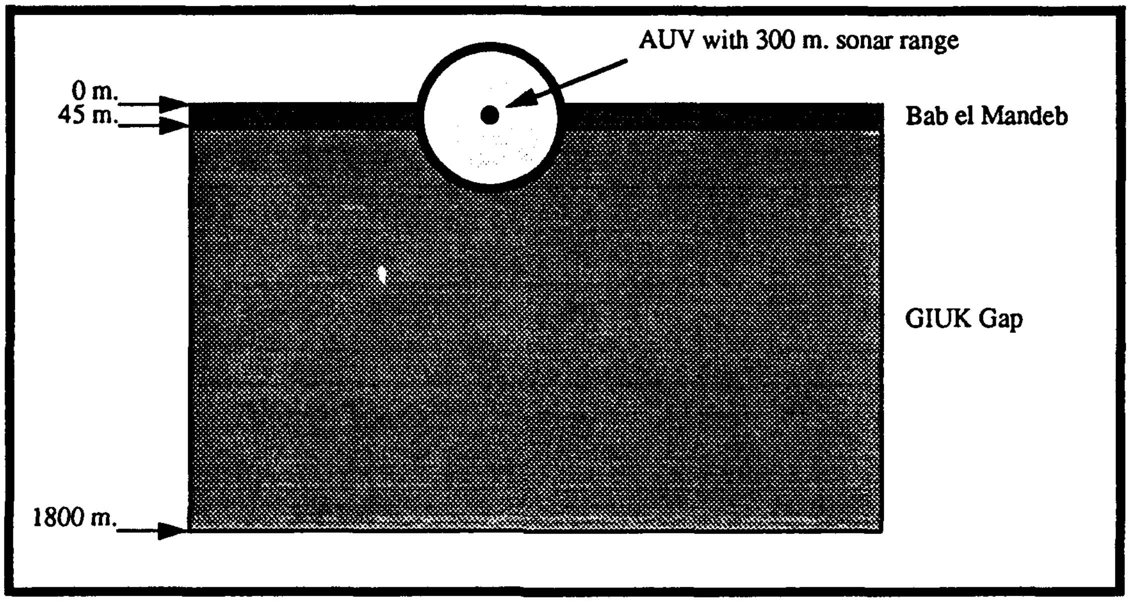


Figure 7: GIUK Gap vs. Straits of Bab el Mandeb Depth Comparison with 300 m.

b. Bottom Topography

The effect of bottom topography on the AUV mine-countermeasure mission will vary depending upon the operating area and target sought. Generally, the bottom topography affects two important mission aspects, AUV navigation and target detection.

While it is important for the AUV to be able to maneuver in close proximity to the ocean bottom, close aboard bottom contour following is not critical for the general conduct of the minefield search mission. As depicted in Figure 8, the AUV may use its sonars to determine whether or not adjacent vertices (hereafter referred to as nodes) are

5. NPS AUV II has 30 meter and a 300 meter range gates.

available. It may then base its choice of paths upon this availability. Should an attainable node be too close to the bottom for safe vehicle maneuvering, the Obstacle Avoidance module (see Figure 3) could provide override control.

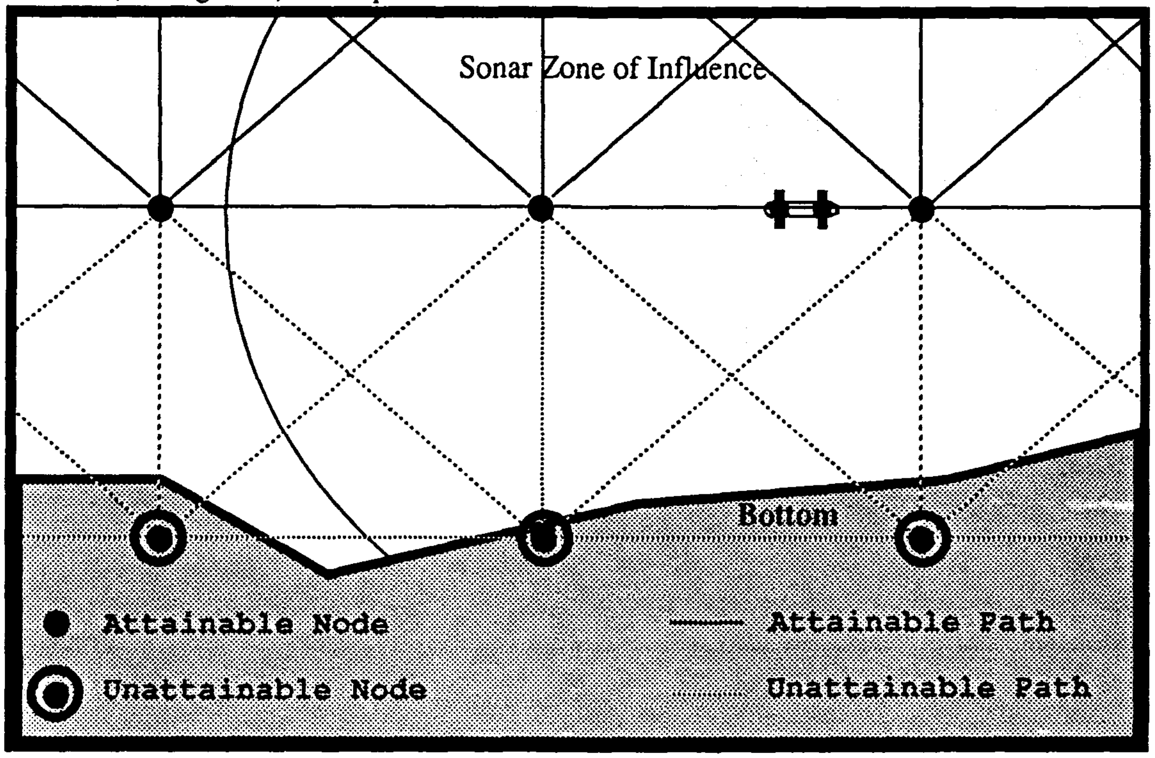


Figure 8: Vertical Depiction of Vehicle Navigation Considerations

Bottom topography also affects the AUV's ability to detect a target. For example, a bottom mine laying on a highly reflective surface such as volcanic rock will be difficult to differentiate from bottom features. A soft, less reflective bottom on the other hand will not interfere with active sonar returns off of the mine.

2. Acoustic Sensor Considerations

a. Passive Acoustic Sensor Considerations

While mines with electronic detection devices do emit noise into the surrounding environment it is of such a small magnitude that it is virtually undetectable against background noise. Passive acoustic sonars are therefore not a primary mine detection tool.

b. Active Acoustic Sensor Considerations

Active sonar is the primary mine detection sensor because it does not rely on target noise emissions. Rather, active sonars emit their own sound wave which reflects off of the target and returns to the sonar receiver. By knowing temperature, pressure and salinity of the water medium a sound velocity is computed. This velocity is applied to the time the sonar pulse takes to travel to and from the target and yields the distance. This, coupled with the direction and beamwidth of the sonar pulse provides the *x,y,z* location of the target.

c. Sonic Layer and Sound Channel Effects

Both active and passive acoustic sound ray paths are affected by changes in the sound velocity of the water. Near the ocean surface, relatively abrupt changes in the sound velocity causes what is known as a sonic layer. Acoustic sound rays traversing this layer bend, creating "shadow zones" and often reducing sonar ranges.

A sound channel is formed when sound is trapped between two mediums. These surrounding mediums can consist of the ocean surface or ocean bottom in conjunction with a changing sound velocity, or sound velocity inversions. The effect is the "trapping" of sound rays, causing extended ranges within the sound channel but shorter ranges and shadow zones outside of the channel.

The existence of these anomalies affects the decision on where to place the AUV for conducting its search. A three-dimensional search algorithm will automatically take care of putting the AUV in the different mediums. The two-dimensional algorithm, however, may leave the areas on the opposite side of the layer or channel from the vehicle inadequately searched. This could be overcome with either the three-dimensional search algorithm or by extending the two-dimensional algorithm into a "two-and-one-half dimensional" search algorithm. This would require that the AUV first search on one side of the layer or channel and then search the other side.

D. AUV EQUIPMENT AND PERFORMANCE CONSIDERATIONS

While most AUVs are severely limited in their current ability to effectively conduct minefield searches, changes in AUV related research are advancing so rapidly that a capable vehicle will likely be available in the very near future. The current state or expected state of the following equipment is an important input into determining how to conduct the minefield search.

1. Depth, Speed and Endurance

a. Depth

This should not normally be a limiting factor for minefield searches since even the deepest mines such as the Mk 60 Captor can be found in only about 370 meters water depth and current vehicles such as DARPA's UUV can dive to about 450 meters.

b. Speed

This is an important factor since it, along wi'h endurance and sensor performance, determines how much area an AUV can search in a given time period. Most AUV's are currently limited to speeds of under 10 knots. This restriction severely restricts the vehicle's ability to search larger areas in a timely manner. By adding the third dimension to a search algorithm the size of the area to be searched is increased dramatically. Therefore, most large area search applications will likely be limited to two dimensions. Another important concern is the effect of speed on the performance of sonars. At the low speeds generally associated with today's AUV's there is little "flow" noise effect on high frequency sonars [Ref. 26]. As AUV speed performance improves, however, this may become another limiting factor in the minefield search.

c. Endurance

This is another important limiting factor in conducting large area searches. Even the DARPA UUV, which has the longest endurance of all AUV's surveyed, has only

24 hours endurance. As in the case of speed limitations, endurance limitations limit the vehicle's ability to conduct large area searches particularly in three dimensions.

2. Sonars

While AUV speed and endurance limit the capabilities of the AUV in large area searches, advancements in sonar technology are dramatically increasing these vehicles' ability to look further with greater definition. The following advancements in sonar technology are important considerations when designing minefield search routines and sonar classification expert systems.[Ref. 26][6]

a. Side-Scan Sonars

This relatively new sonar is the most commonly used for small object detection, location, and classification. It is most effectively used against targets lying on or near the bottom. These sonars typically have maximum detection ranges of under 600 meters although some like the Sea Mark I have ranges out to 2500 meters. Generally, side scan sonars operate at an altitude of 10 to 20 percent of their maximum detection range above the bottom. A typical side-scan sonar profile appears in Figure 9. It is important to note that sonar coverage is generally abeam of and below the AUV and that little obstacle avoidance sensing is provided.

b. Obstacle-Avoidance Sonars

These high frequency sonars are typically used for obstacle detection and avoidance including terrain avoidance. Included in this category are the sonars currently installed on the NPS AUV II.

E. NAVIGATION CONSIDERATIONS

Precise AUV positional information is critical in the minefield search problem. It is necessary for both getting the vehicle where it is supposed to be and for returning mine and

6. [Ref. 26] provides excellent descriptions of side-scan, forward-looking and down-looking sonars.

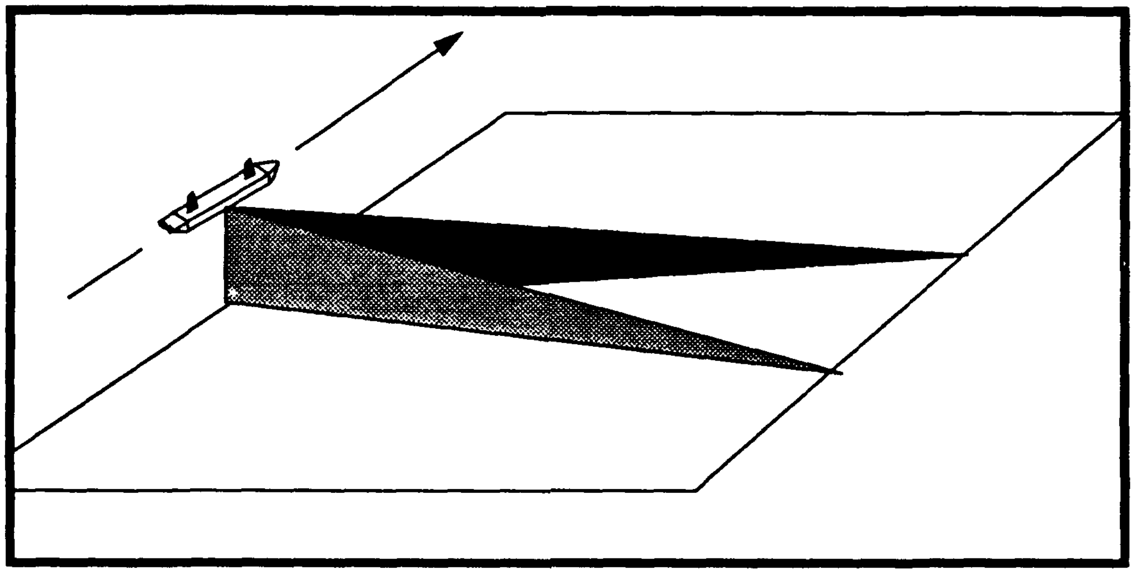


Figure 9: Side-Scan Sonar Profile

other obstacle positional information to the parent ship. Inertial navigation positional information is highly accurate over short durations but degrades in proportion to the time between navigational fixes. Global Positioning System (GPS) holds the most promise for providing highly accurate fixing data. Current non-military systems are capable of fixing the AUV's position to 100 meters in real time and down to 2 meters using differential post processing. Military systems are even more capable.[Ref. 27]

Another means of fixing the AUV's position is through bottom topography. Bottom contour following is a navigational method practiced by both ships and submarines. It is also possible in AUV's. Another means would be for the AUV to fix its position using known bottom features. These features could be extracted using an expert system similar to that proposed in Chapter VI.

IV. MINEFIELD SEARCH PROBLEM SOLUTION

A. MINEFIELD SEARCH PHILOSOPHY

Traditional robot searches use algorithms based on variations of search routines involving a start, a goal, and obstacles which obstruct the vehicle's path. The minefield search also has a start point and obstacles but it differs considerably in its high level goal. In the minefield search algorithm, the goal is the complete coverage of the minefield with the vehicle's sensors (reflected in the high-level tier of Figure 4). This means that the vehicle must have some method of determining which parts of the search area have been adequately searched and which have not. The vehicle must be able to efficiently maneuver to those areas yet to be searched (intermediate-level tier of Figure 4). Also, the vehicle is likely to complete its mission in different locations depending on the type of obstacle patterns it encounters. It is apparent that the typical start to goal type search routines are unsatisfactory for providing a solution on the highest level of a search of this type.

This chapter presents an original minefield search algorithm which guides the vehicle in thoroughly and systematically searching unknown terrain. It is pure in nature in that previous knowledge of the search area is not required. It follows the general area search hierarchy presented in Figure 4 in its philosophy and takes advantage of the efficiencies associated with the more traditional search algorithms in completing its goal. Both the two-dimensional and three-dimensional algorithm versions are discussed.

B. MINEFIELD WORLD REPRESENTATION

1. *A priori* World Knowledge

According to Chappell, "A critical component of the guiding intelligence of an autonomous vehicle is the system's computational model of its physical environment. An autonomous entity with no model of its environment is severely limited." [Ref. 24] While this may be true for vehicles maneuvering in very close proximity to a dense field of obstacles it is not a requirement for vehicles which maintain significant safe-standoff ranges. A significant characteristic of the minefield search algorithm is that it does not require such a detailed *a priori* map of the world. Rather, it only requires definition of the search area boundaries and the spacing between nodes.

2. Search Area Shape

An important consideration for the data structure of the world is the shape of the search area. Large, open-ocean search areas could best be defined with two or three-dimensional, rectangular shaped boundaries. This is because there are no large obstacles to obscure parts of the search area. This choice may not do well for odd-shaped choke points or for areas with large variations in bottom topography. These types of areas may best be defined by a more flexible structure which more efficiently defines the boundaries of the vehicle's search area. Figure 10 shows such contrasting search areas. These arguments

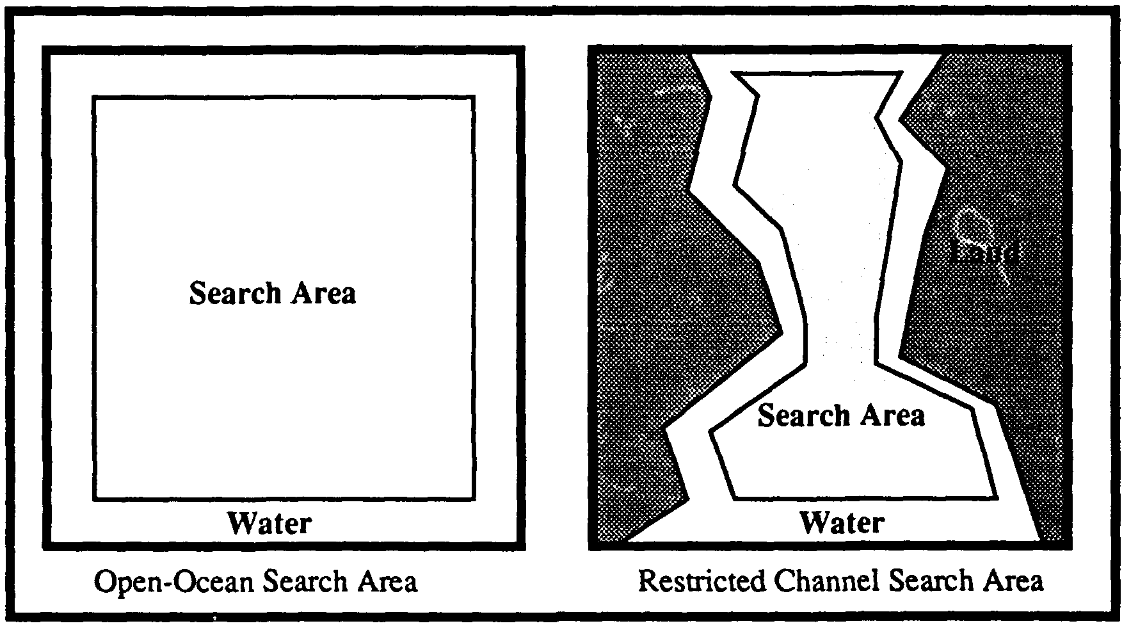


Figure 10: Contrasting Open-Ocean and Restricted Channel Search Areas

would hold true if *a priori* world knowledgs were essential for vehicle navigation. It would also be important if the AUV's computer memory was so small that storing a structure which included areas known to be unattainable would push memory capacity limits. The minefield search presented here assumes that the additional memory required to store such unattainable areas is so small as to be insignificant. It also eliminates the need for providing precise *a priori* boundary definitions including bottom topographical boundaries because it effectively excludes inaccessible areas within the algorithm. Therefore, the search area may be defined in a very generic manner as in Figure 11.

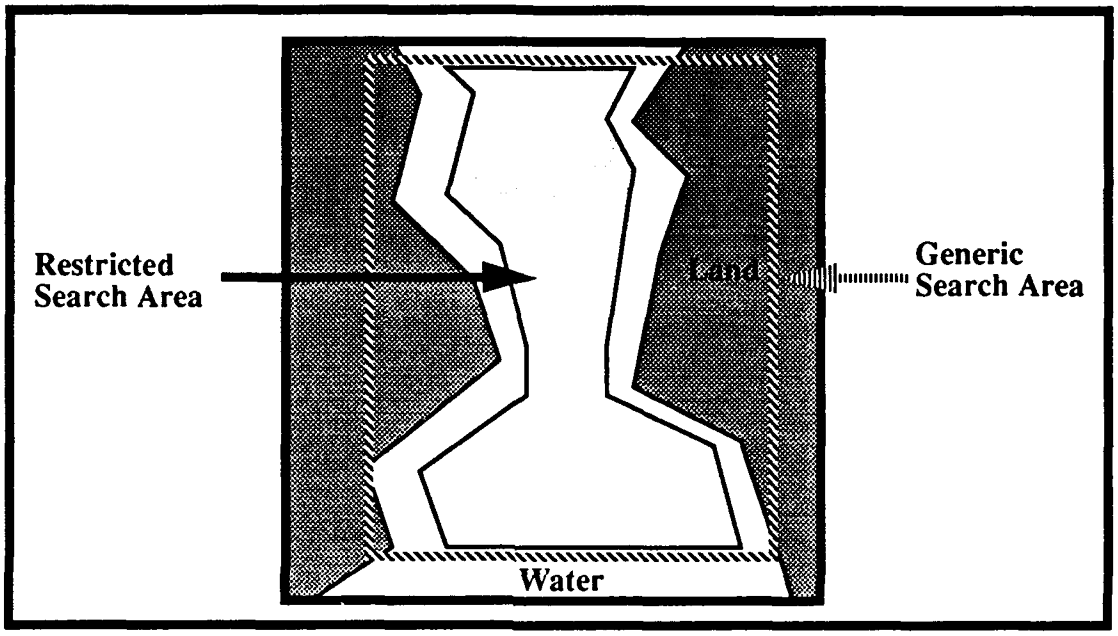


Figure 11: Generic Search Area Dimensions

3. Graph Representation of the World

a. *Fundamental Graph Structure*

The graph-based representation method was chosen as the structure for the minefield search world model. Within this representation, the world is fundamentally defined by two-dimensional or three-dimensional rectangular graphs shown in Figure 12. These graphs are locally defined by fully connected adjacent nodes *N*. The nodes are defined by the Cartesian coordinates *(x,y)* in two-dimensions and *(x,y,z)* in three dimensions, and are connected by non-directional edges. The distance between nodes and thus the edge length is determined by either the human operator or by the Mission Planner in module 1 of the dataflow diagram. This distance will be a function of the sonar's

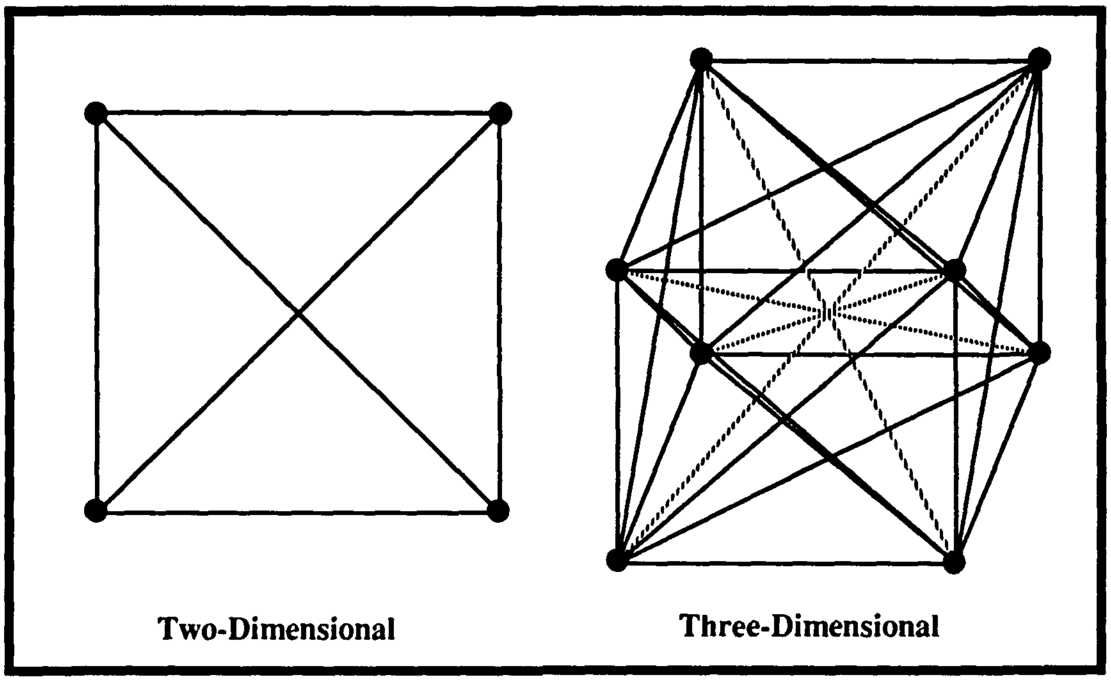


Figure 12: Basic Building Blocks for Two and Three-Dimensional Graph Representation

capability, water temperature, water pressure, water salinity, and the desired probability of detection (POD).

b. Composite Graph Structure

Both the two and three-dimensional graphs are created by piecing together the fundamental structures. Figure 13 shows four two-dimensional fundamental graph structures pieced together. Note that with the AUV placed at the center there are eight possible adjacent nodes to which the vehicle can travel.

Figure 14 is the result of piecing together four fundamental three-dimensional structures. With the vehicle placed in the center of the graph, there are 26 possible adjacent nodes to which the AUV can travel. In the minefield search algorithm, up (U) and down (D) have been excluded from consideration because of the assumption that this particular AUV cannot travel vertically. Therefore, this AUV is limited to traveling to 24 adjacent nodes.

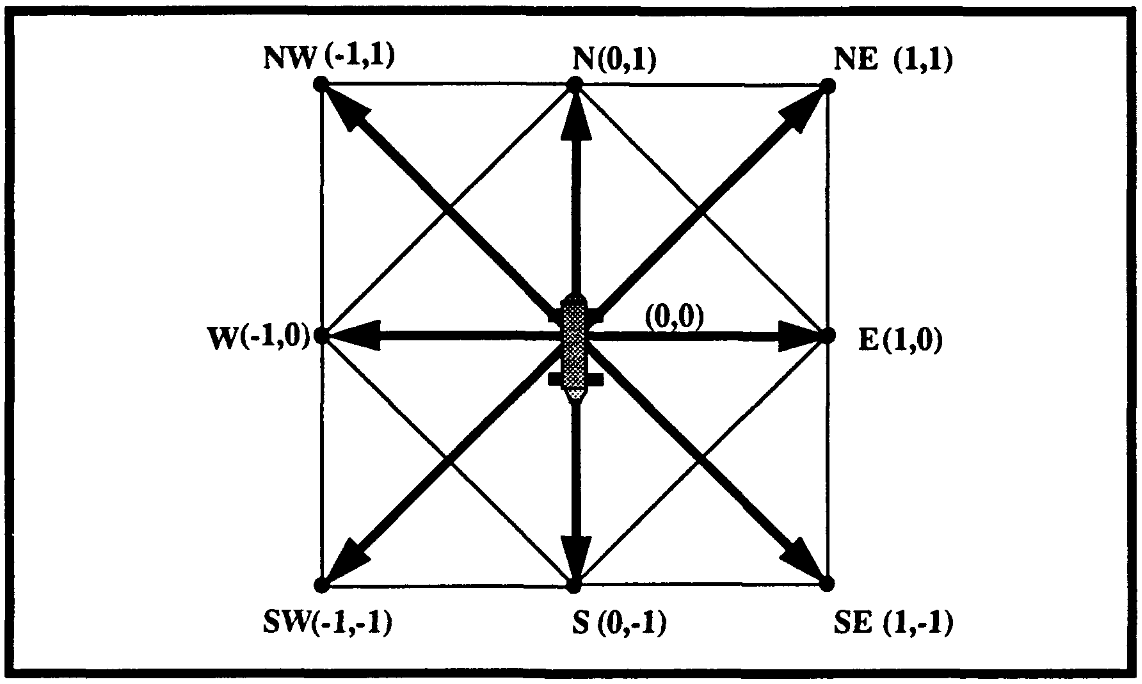


Figure 13: Two-dimensional model provides eight possible directions to traverse.

An alternate method of visualizing the graph structure is to assume that each node is located in the center of the squares (two dimensions) or cubes (three dimensions) which are hereafter referred to as blocks as seen in Figure 15.

4. World Structure

These fundamental structures are pieced together until the entire search area is covered thus forming the search world structure. Note that number of possible nodes to which the vehicle can travel is reduced when the AUV is at a world boundary node.[1]

5. Sub-World Structure

To complement a high-level search plan G. Dudek, et al., suggests that the world can be divided into subgraphs or sub-areas and that exploration be confined to these compact regions until they have been fully searched [Ref. 18]. This concept is very significant when searching a minefield. If a search algorithm does not use a sub-area

1. This is a node located on the corner, outside edge or face of the world search area.

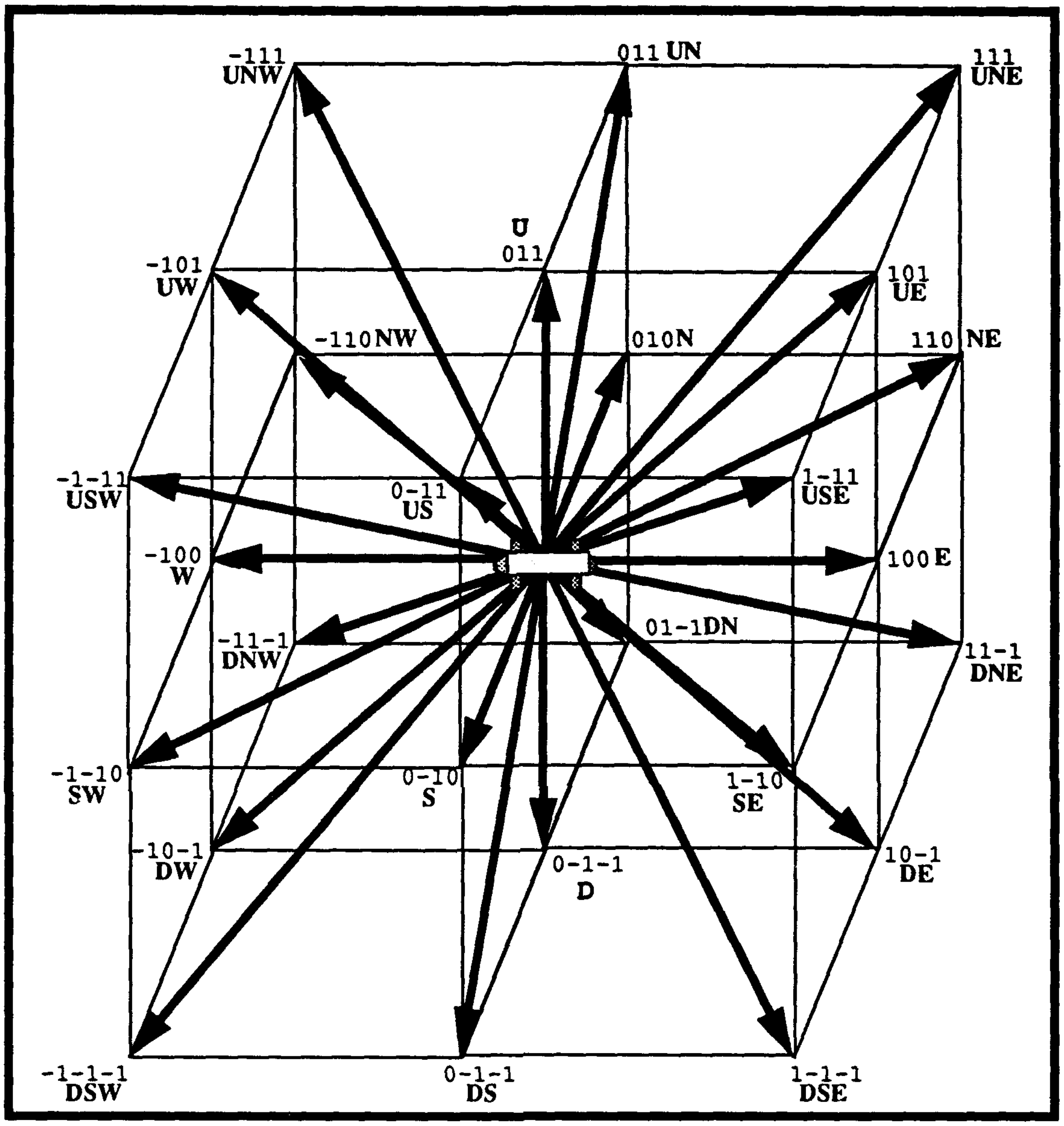


Figure 14: Three-dimensional model provides 26 directions to traverse.

strategy then it will generally leave many gaps in its search when obstacles are encountered and may not get back to search these gaps until late in the mission. If the vehicle is unable to complete its mission then these gaps are left unsearched. Such somewhat spurious gaps may prevent the parent ship, submarine, battle group or amphibious task force from advancing into the minefield. On the other hand, if the AUV has the priority to search entire sub-areas prior to moving on to new sub-areas and these sub-areas are searched in a

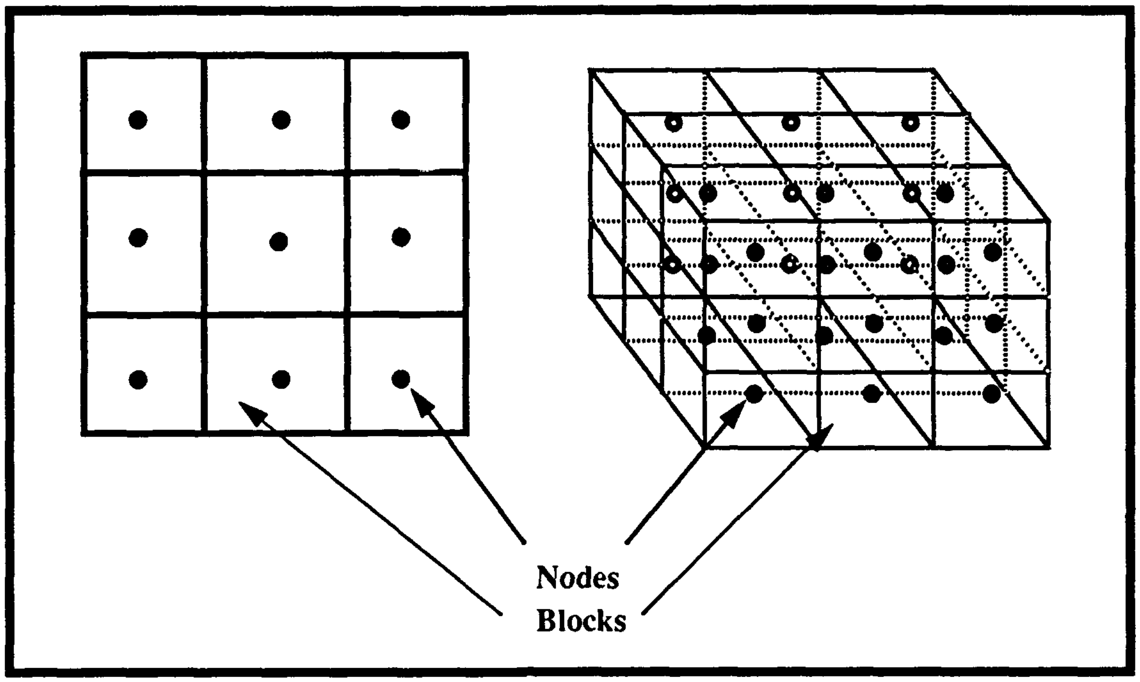


Figure 15: Alternate Representation of Basic Building Blocks for Graph Representation

systematic fashion then fewer gaps are left in the search of the minefield. This would allow the ships to advance safely to the boundaries of the AUV's successful search. This distance advanced may in many cases be enough for the ships to begin conducting their primary missions.

a. Two-Dimensional Sub-Area Construction

Figure 16 shows how the two-dimensional minefield search area for this thesis is subdivided into 49 sub-areas numbered in order of priority, with 81 nodes/blocks to each subarea for a total of 3,969 nodes/blocks. The number of sub-areas and the number of nodes/blocks in the *x* and *y* directions and the track spacing are defined by global variables in order to flexibly meet the user's needs. A blow-up from a discrete graphic simulation mission replay shown in Figure 17 shows how a vehicle using the minefield search routine with prioritized sub-areas systematically covers every available node/block in the available sub-areas. In contrast, Figure 18 shows how the vehicle using the same minefield search routine without prioritized sub-areas leaves coverage gaps.

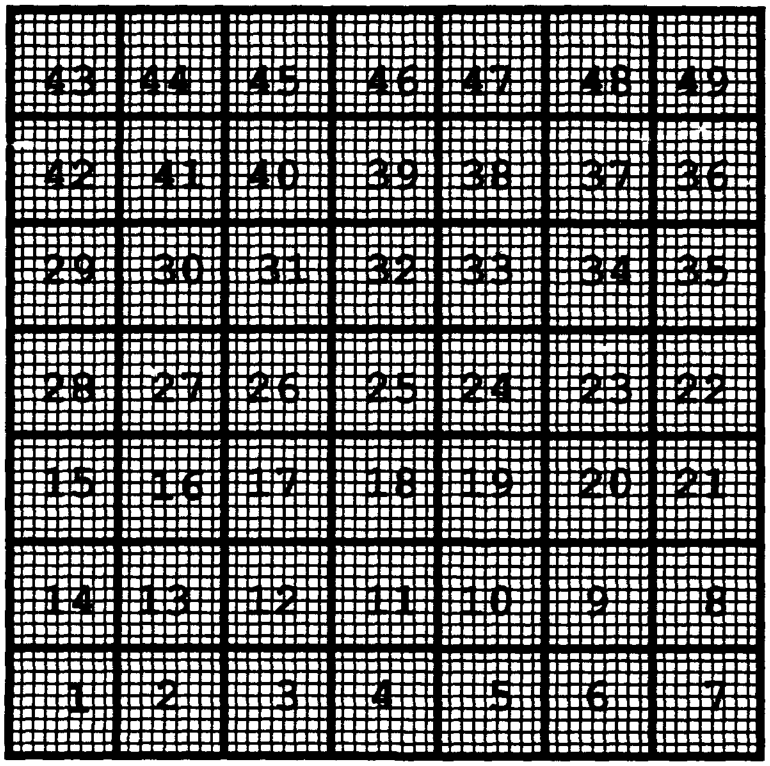


Figure 16: Two-Dimensional Search Area Sub-Divided into Prioritized Sub-Areas

b. Three-Dimensional Sub-Area Construction

The choice of how to organize sub-areas in the three-dimensional world follows a similar philosophy to that used in the two-dimensional organization. It remains important that portions of the search area be thoroughly investigated prior to moving on to other areas. Figure 19 shows how the three-dimensional sub-areas are prioritized. Note that slices of ocean in the vertical are searched prior to advancing down-track. As in the two-dimensional program, the number of sub-areas and nodes/blocks in the *x*, *y* and *z* directions and the track spacing are globally defined for user flexibility. For evaluation purposes, this thesis uses 75 sub-areas, five in each of the *x* and *y* directions and three in the *z* direction. Each sub-area consists of five nodes/blocks in both the *x* and *y* directions and three in the *z* for a total of 5,625 nodes/blocks.

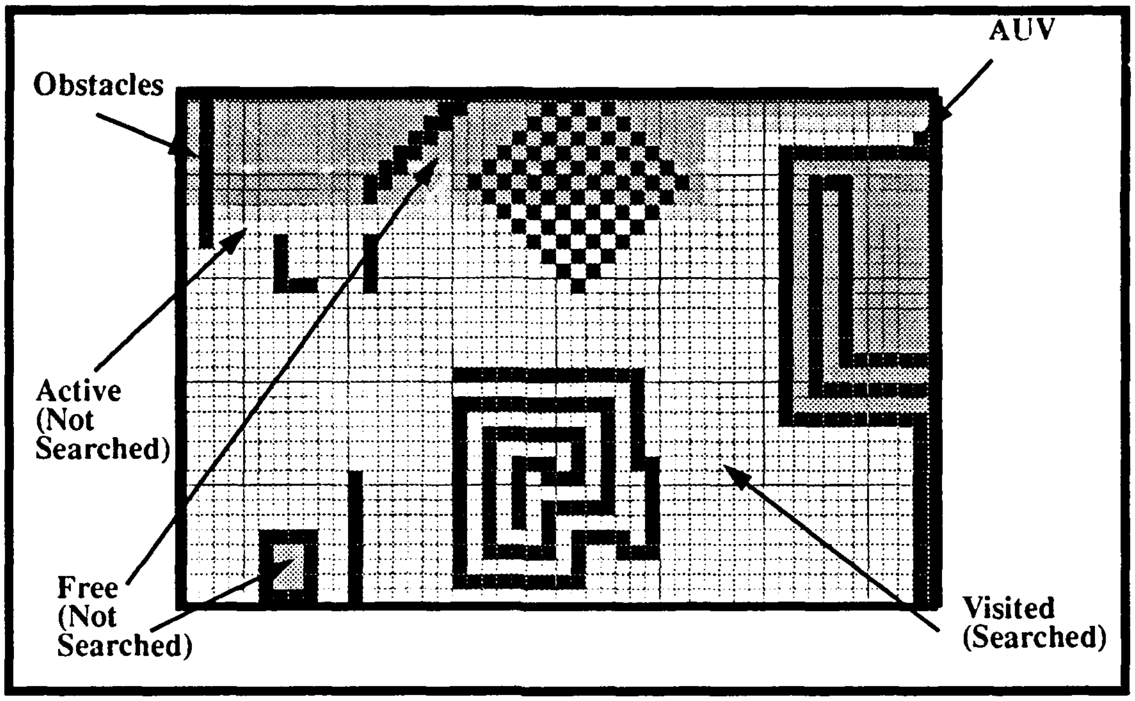


Figure 17: Prioritized Sub-Area Minefield Search

6. World Data Structure

The minefield search world is defined as a two or three-dimensional array of nodes/blocks. The number of nodes/blocks in the *x*, *y* and *z* directions and the track spacing are defined by global variables to allow the user to search an area of any desired size.

Each node/block is defined by its *x*,*y*,*z* position, its sub-area number, and its state. The state of the node/block is an important determinant of the actions available to be taken on that node/block. There are two varieties of states, primary and secondary. Primary states are mutually exclusive amongst each other and are used in the high-level minefield search algorithm, the intermediate-level path planning algorithm and the graphic simulator for display purposes. Secondary states are not mutually exclusive amongst primary or secondary and exist only for the purpose of either intermediate-level path planning or graphic simulator analysis. The possible primary states are defined as follows:

- FREE. This indicates that the block has not been searched by the AUV. This also means that the corresponding node has not been analyzed or visited by the AUV.

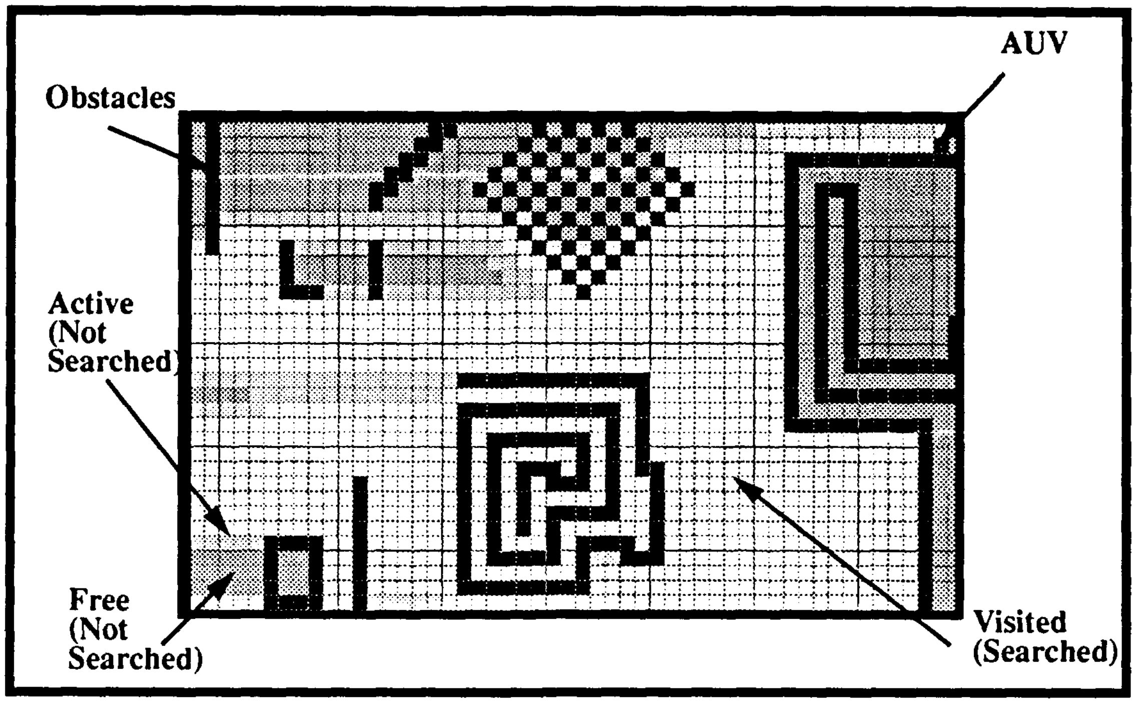


Figure 18: Non-Prioritized Sub-Area Minefield Search

- OBSTACLE. This state indicates that the node is unattainable from at least one incoming edge. Therefore, an obstacle is considered to exist at that node. Obstacle takes on a broad definition in this context. It includes mine, ocean bottom, sea-mount, ship, or any other obstruction.

- AUV. This state indicates that the AUV has most recently been at this node and has yet to achieve the next node.

- ACTIVE. An ACTIVE node has been analyzed by the AUV's sonar and determined to be attainable [2](not OBSTACLE). The corresponding block has not been thoroughly searched nor has the AUV visited the node/block.

- VISITED. This node has been visited by the AUV and its corresponding block has been thoroughly searched.

2. An attainable node is not only one which is not obstructed by an obstacle but also one in which adjacent nodes are not all obstructed by obstacles.

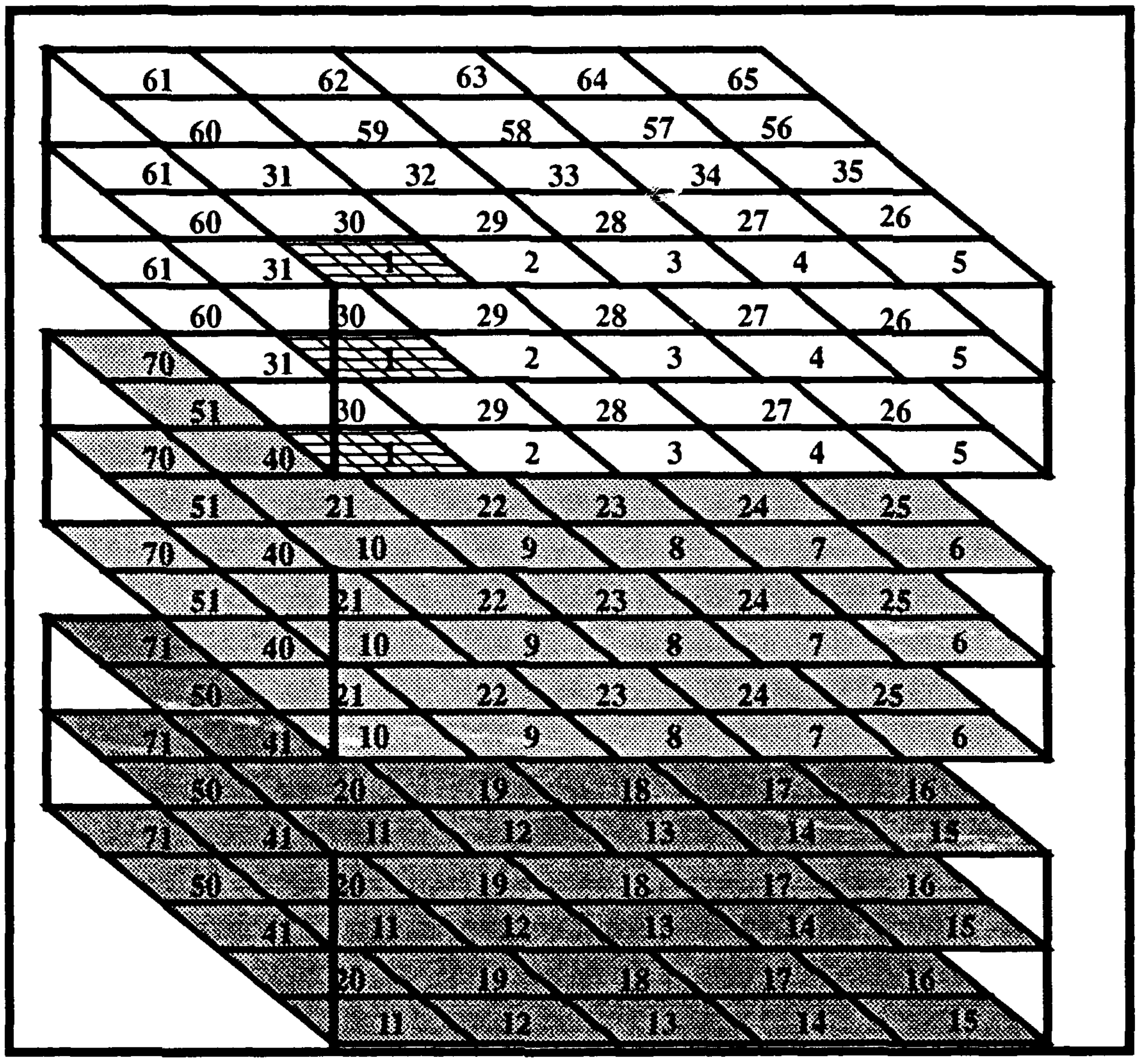


Figure 19: Three-Dimensional Search Area Sub-Divided into Prioritized Sub-Areas

The secondary states are ADJACENT and ASPATH. The former represents those adjacent nodes being evaluated by the AUV's sonar. The latter indicates that the node is on the path from the AUV's current position to its goal as defined by the intermediate-level path planning algorithm.

C. MINEFIELD SEARCH ASSUMPTIONS

1. Sonar Coverage Assumptions

There are some important sonar coverage assumptions which were used when developing the minefield search algorithm. First, the virtual AUV has a much more sophisticated sonar suite than that found on the NPS AUV II. This assumption was necessary because of the very restricted coverage provided by the NPS AUV II's four transducers. Rather, the presently fictitious AUV used for this algorithm has the type of nearly spherical coverage which would be possible on a vehicle using a towed array sonar or a vehicle covered by the "mechanical hydrophones" discussed earlier.

2. Track Spacing Assumptions

The desired vehicle track spacing [3] is a function of the sonar's capability, environmental variables, and the desired probability of detection. The minefield search algorithm bases the track spacing on the vehicle's ability to determine if an adjacent node is attainable from the AUV's current node. Figure 20 shows an acceptable track spacing based on sonar range. Note that this choice of spacing ensures that the AUV can use sonar to analyze all adjacent nodes to determine if they are attainable. This also ensures that the block in which the AUV is located is searched with a very high POD. It does not, however, ensure that adjacent blocks are thoroughly searched.

3. AUV Characteristics Assumptions

The first assumption is that the AUV will not be powered and balanced to allow it to traverse vertically (e.g., $N(x_n,y_n,z_n)$ to either $N(x_n, y_n, z_{n+1})$ or $N(x_n,y_n,z_{n-1})$).

The next assumption is that the AUV's turn radius is small enough in comparison to the track spacing that it is able to maneuver within close proximity of a node for the purposes of doing a 180^{o} reversal of course to exit "blind alleys."

3. Track spacing is defined as the distance between two possible vehicle paths which are parallel and adjacent.

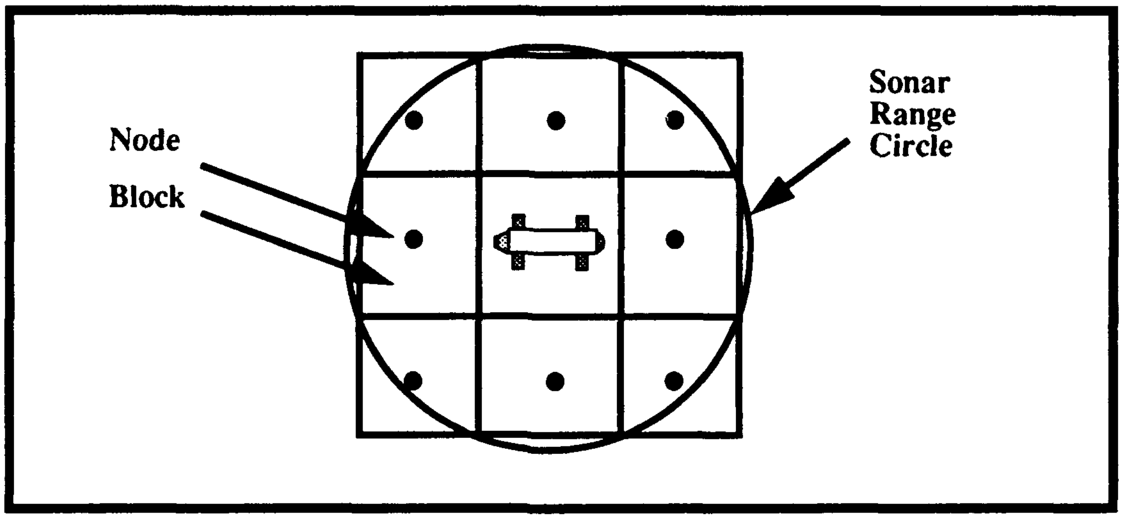


Figure 20: Track spacing requires sonar be able to determine adjacent nodes are attainable (free of obstacles).

Navigational information is assumed to be perfect.

D. SUMMATION OF MINEFIELD SEARCH REQUIREMENTS

The following requirements dictate the conduct of the minefield search algorithm in static obstacle fields:

- The AUV must search all attainable blocks in the defined search area.
- The AUV must be able to commence its search from any node not obstructed by an obstacle.
- The AUV must attempt to optimize the search by ignoring nodes which are unattainable.
- The AUV must achieve coarse maneuvering around obstacles.[4]
- The AUV must handle stove pipe[5] and blind alley[6] situations.

4. This includes the ocean surface, the ocean bottom, and other features such as sea-mounts.
5. A stove pipe is a vertically oriented object which is open in the center. It is similar to a stove pipe found on old wood burning stoves. If there is only one vertical column of nodes through the center of the stove pipe, then a path through the stove pipe is unattainable.
6. Blind alleys are horizontally oriented stove pipe like objects.

E. HIGH-LEVEL MINEFIELD SEARCH ALGORITHM

1. Ladder Search Theme

As stated previously, in the high-level search strategy it is desirable to systematically and thoroughly search areas of ocean prior to moving on to other areas. While the sub-areas are searched in a prioritized manner[7] based on the ladder search theme, the blocks within a sub-area are not prioritized. They are, however, searched in a similar systematic fashion.

Within the sub-areas it is desirable to maintain as much of a ladder type search methodology as possible. In a sub-area free of obstacles, the search would proceed from block to block exactly as in Figure 21. Problems arise, however, when obstacles are introduced into the minefield. As can be seen in Figure 22 the vehicle is no longer able to maintain ladder search integrity and gaps in coverage develop (Figure 17). While it is impossible to maintain ladder search integrity under these circumstances, heuristics may be employed which maintain the ladder search theme as much as possible.

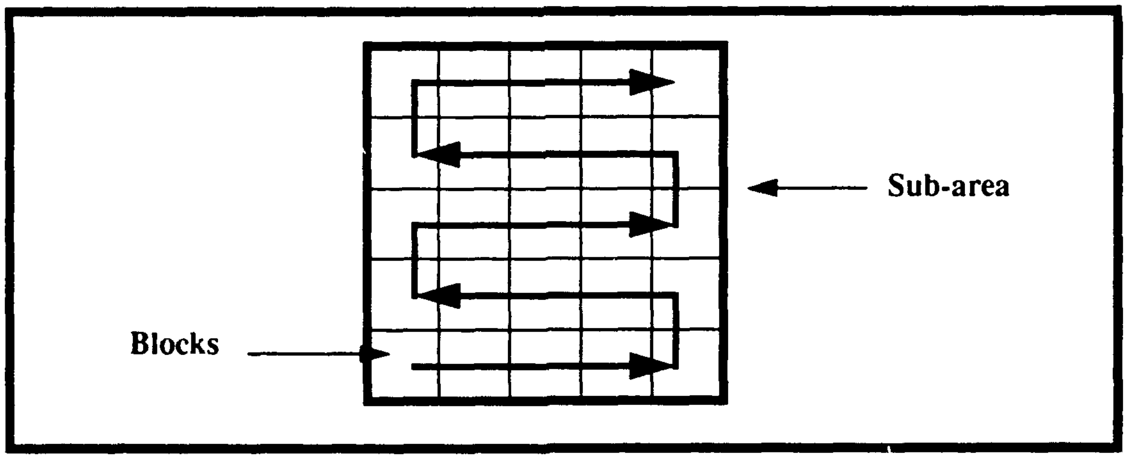


Figure 21: Two-Dimensional Ladder Search Methodology within a Sub-Area

7. Sub-areas are prioritized in decreasing priority with sub-area one being highest priority.

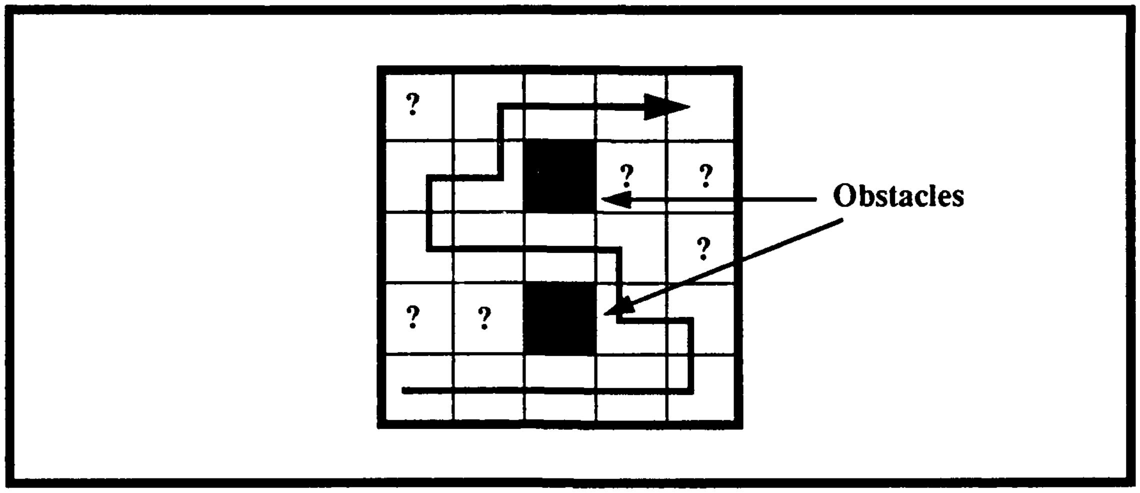


Figure 22: Ladder search coverage falters with introduction of obstacles.

2. AUV Start/End Location

An important feature of the minefield search algorithm is that the starting point from where the AUV begins its search can be anywhere within the world boundary.[8] The conduct of the algorithm is independent of this starting position. The AUV will still proceed to the higher priority sub-areas as they become known to the vehicle.

The end location for the AUV is also variable and depends on the conduct of the search. If a specific pick-up point were required it could be easily attached to the minefield-search algorithm as the last point to be visited. The intermediate-level path planning would then determine an appropriate path to the end point.

3. Data Structure for Tracking Blocks which Need to be Searched

As the AUV passes by adjacent FREE nodes, the states of these nodes are changed from FREE to ACTIVE. This indicates that each node is attainable and its corresponding block must be searched. While this information could be kept in the world array, it would be inefficient since the algorithm would have to scan the entire array to determine which nodes are ACTIVE and their relative priorities. A much better data structure for keeping

8. It is assumed that the vehicle will be launched at a node within the confines of the world boundaries and that the starting node is not obstructed by obstacles.

track of this information is the linked list. While a linked list is not quite as efficient as a heap, it is much more straight forward to implement in this application.

The priority of nodes to be visited is determined on two levels. The highest and most important level bases its node selection criterion on the priority of the sub-area in which the ACTIVE node resides. Given more than one ACTIVE node in the same sub-area, the lower level bases the selection criteria on heuristics to be discussed in a later section.

Selection of ACTIVE nodes for searching based on sub-area priority is a very straight forward matter and is independent of the AUV's current location. The sub-areas which have ACTIVE nodes can therefore be kept in a sorted linked list with the highest priority "active" sub-area at the top of the list.

Selection of ACTIVE nodes for searching based on the lower level heuristics is AUV position dependent and therefore varies with the vehicle's movements. Thus, a sorted list of ACTIVE nodes does not serve any productive purpose.

A preferred hybrid of the linked list which accommodates these facts is the list of lists. This hybrid is implemented in the minefield search to track ACTIVE nodes and is referred to as the "visit list"(Figure 23).[9]

Each active sub-area retains a record in the sorted linked list (referred to as the "sub-area list") with a pointer to an unsorted linked list of ACTIVE nodes (referred to as the "node list"). If a node changes its state to ACTIVE the sub-area list is searched for a match to the node's sub-area. If one does not exist, an active sub-area record is created and inserted according to its priority. A record for the ACTIVE node is also created and inserted at the head of the active "node list" corresponding to that sub-area.

If an ACTIVE node in the same sub-area is already listed, the node record is simply inserted at the head of the list corresponding to the already existing active sub-area record. As the last ACTIVE node record is removed from a particular node list, the corresponding sub-area record is also removed.

9. The visit list consists of a sorted active sub-area list pointing to unsorted active node lists.

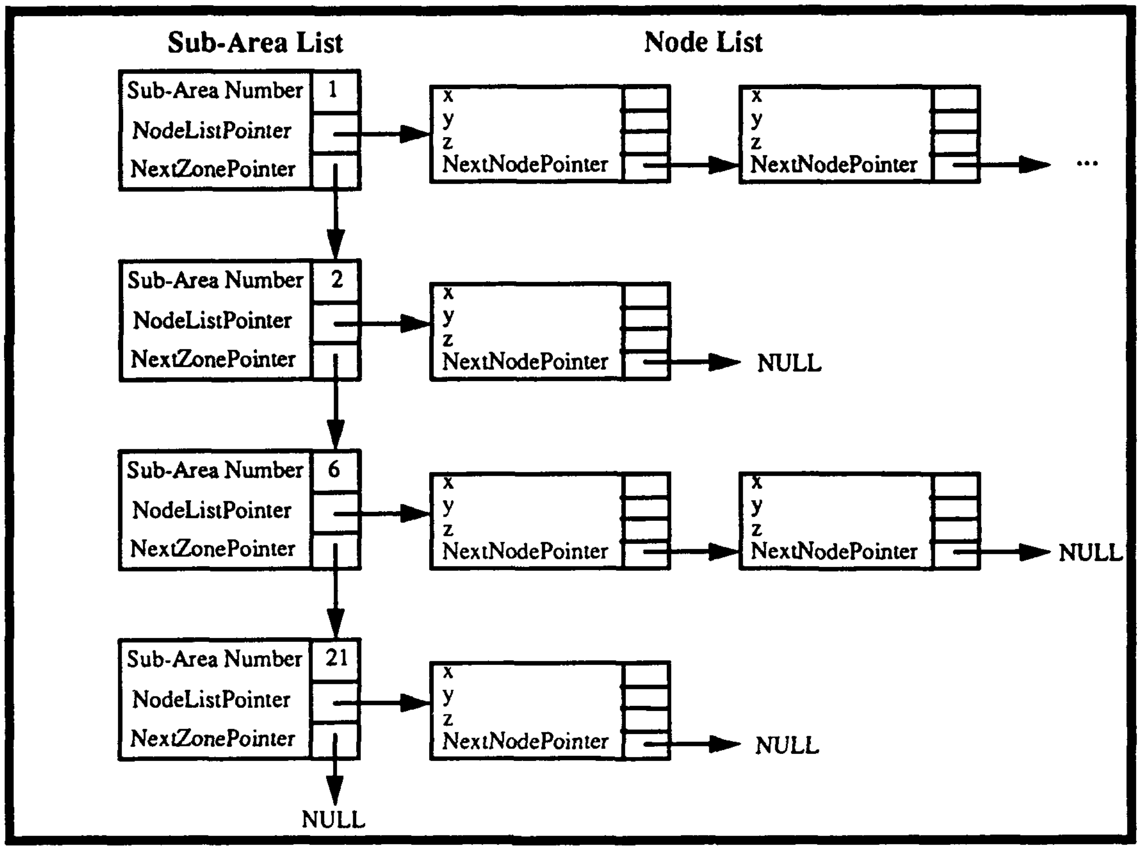


Figure 23: The "Visit List" List of Lists

4. Algorithm Flow

The flow diagram in Figure 24 gives the higher level process for conducting the minefield search.

a. Analyzing Nodes Adjacent to the AUV

When the AUV is placed at the start node, that node's state is annotated AUV. The vehicle then searches the block in which it is located and concurrently analyzes its adjacent nodes. Figure 25 diagrams this procedure. The adjacent nodes can be exhaustively analyzed in any order. If the node is blocked by an obstacle the node is labeled OBSTACLE. If a node is not an OBSTACLE, is within the world boundaries, and has the state FREE, than it is added to the visit list. So long as the visit list is not empty there are

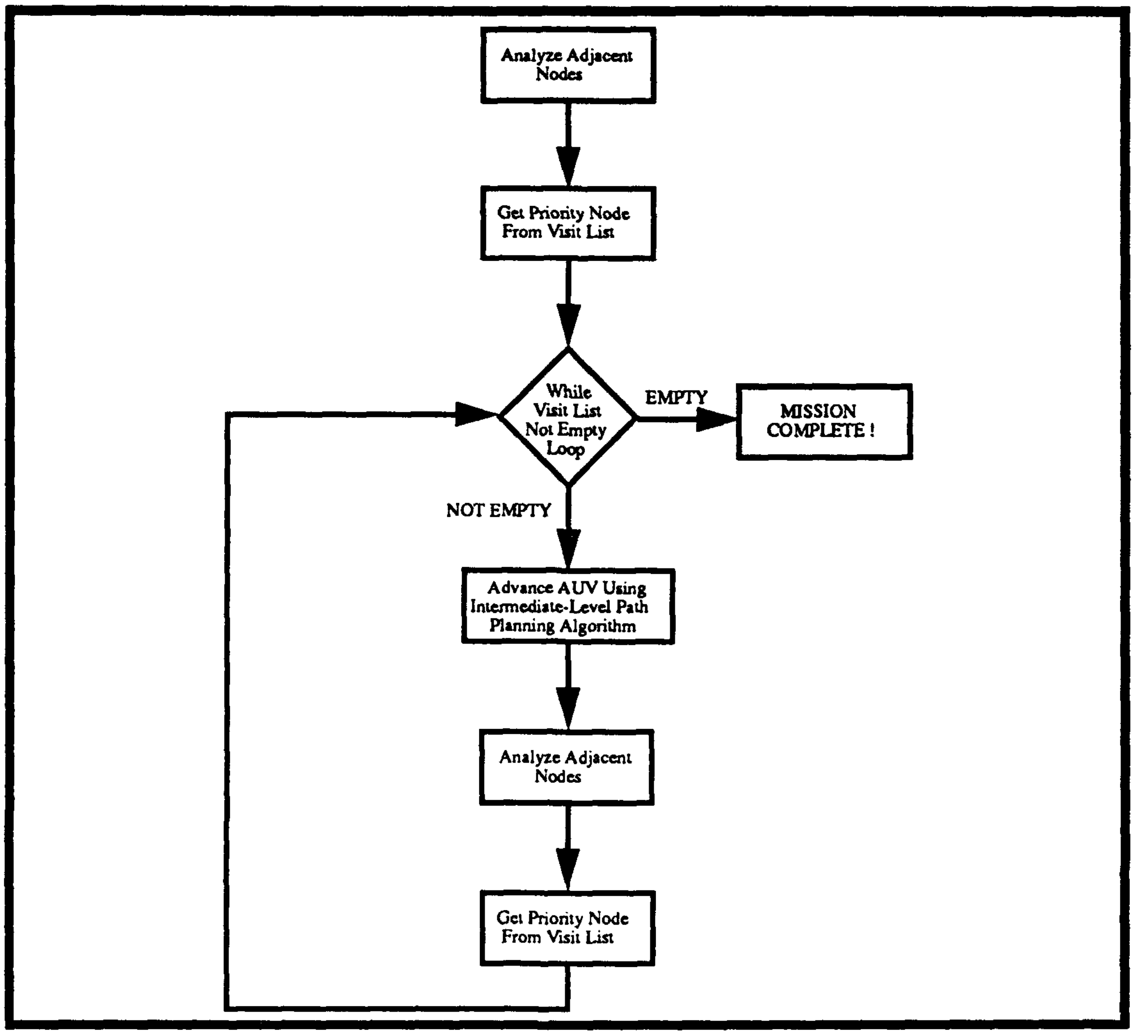

Analyze Adjacent Nodes
Get Priority Node From Visit List
While Visit List Not Empty Loop
EMPTY
MISSION COMPLETE !
NOT EMPTY
Advance AUV Using Intermediate-Level Path Planning Algorithm
Analyze Adjacent Nodes
Get Priority Node From Visit List

Figure 24: Minefield Search using Prioritized Sub-Areas Algorithm

blocks which need to be searched. Once the visit list returns NULL, all attainable blocks will have been searched and the mission is deemed completed.

b. Determining Priority of Nodes/Blocks to be Visited

After the nodes adjacent to the AUV have been analyzed the priority node[10] is retrieved from the visit list (see Figure 26). Because the sub-area list portion of the visit list is always prioritized, the first record in the sub-area list points to the node list from

10. The priority node equates to the AUV's interim goal to which it conducts its intermediate level path planning search.

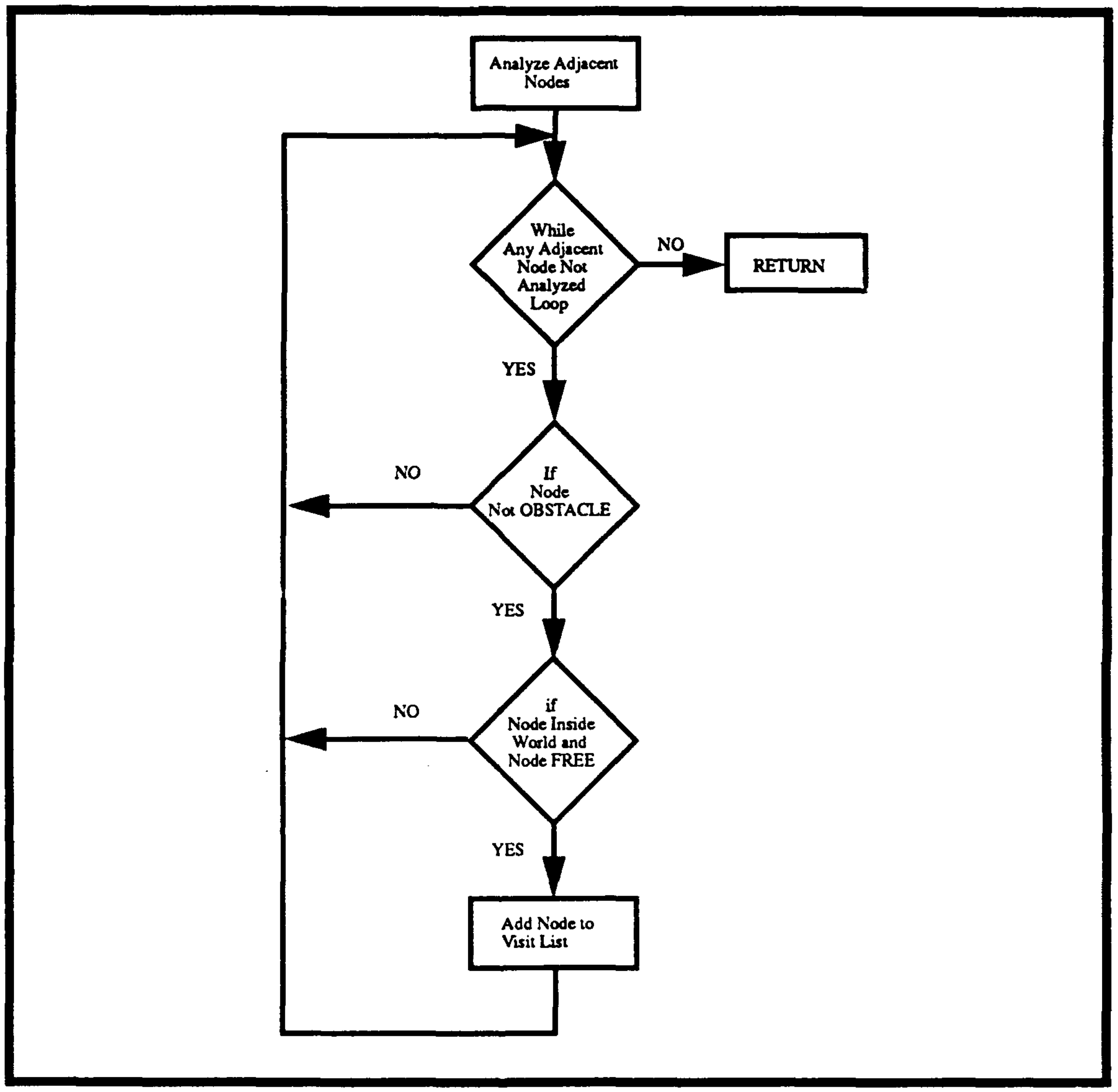
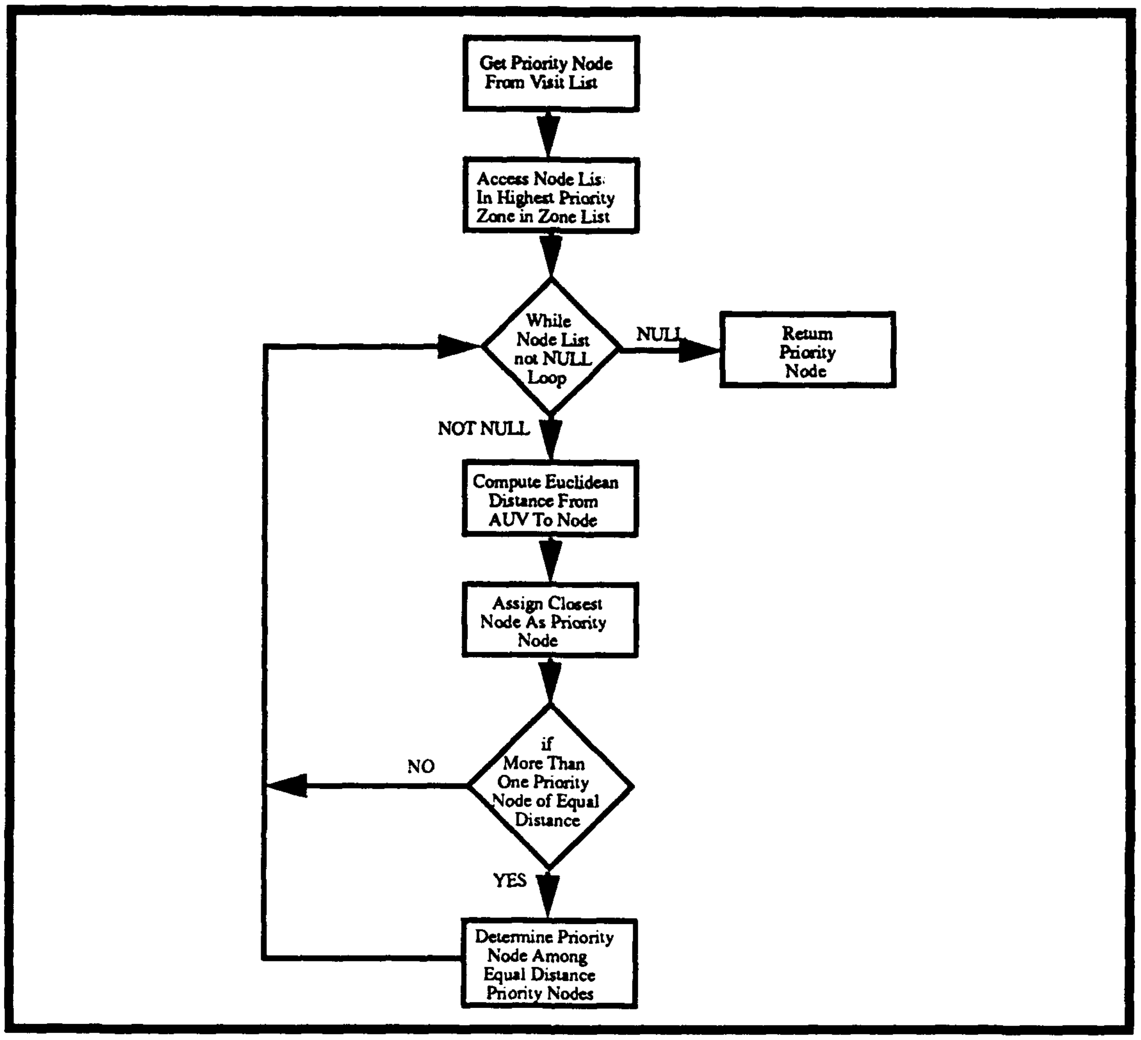


Figure 25: Algorithm to Analyze Adjacent Nodes and Add to Visit List

which the priority node will be chosen. It is necessary to traverse this entire node list, comparing the selection criteria values of one node to the next. A "best node" pointer always points to the node which is the current highest priority node.

The primary criterion for selecting the priority node within a sub-area is the Euclidean distance from the AUV. This criterion was chosen because it favors the four cardinal points[11] over diagonal points in both the two and three-dimensional searches when

11. The vertical cardinal points are not considered in this three-dimensional search.



Get Priority Node From Visit List
Access Node List In Highest Priority Zone in Zone List
While Node List not NULL Loop
NULL
Return Priority Node
NOT NULL
Compute Euclidean Distance From AUV To Node
Assign Closest Node As Priority Node
if More Than One Priority Node of Equal Distance
NO
YES
Determine Priority Node Among Equal Distance Priority Nodes

Figure 26: Priority Node Based on Distance Criterion

choosing amongst adjacent nodes. This minimizes the number of diagonal traverses which have a distance ratio of about 1.4:1.0 in comparison to cardinal traversals. It also helps the AUV keep to the ladder search theme.

There are numerous possibilities for a tie between closest points of equal distance. In the two-dimensional search there may be as many as four adjacent nodes which tie under this criterion. In the three-dimensional search there may be as many as twelve adjacent nodes. There are even more possibilities for non-adjacent nodes. Therefore a secondary priority node selection method based on relative node location heuristics was chosen. If the node being compared to the current best node is of equal distance, relative location heuristics are used to determine which node is actually the best node. The flow chart in Figure 27 outlines these heuristics. Basically, they favor nodes at the same depth

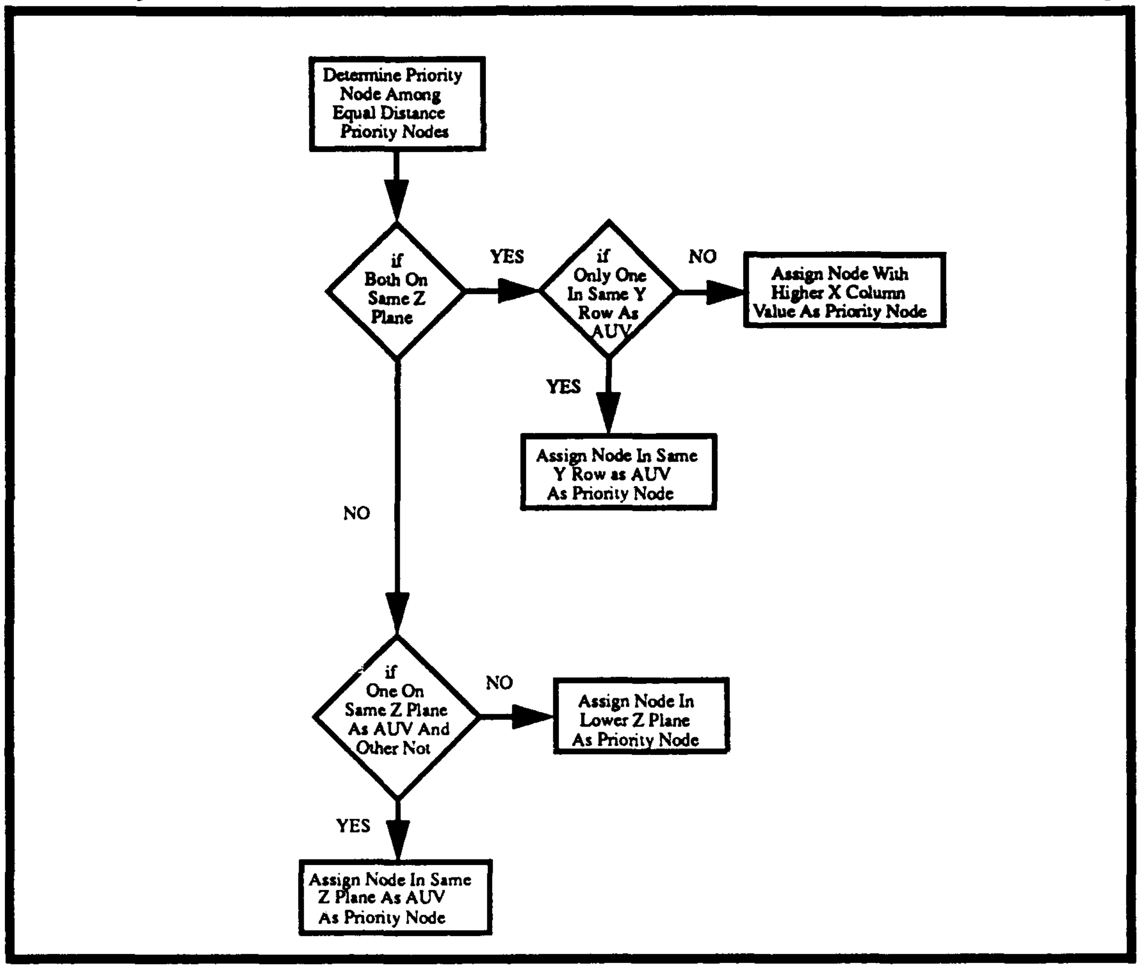

Determine Priority Node Among Equal Distance Priority Nodes
if Both On Same Z Plane
YES
if Only One In Same Y Row As AUV
NO
Assign Node With Higher X Column Value As Priority Node
YES
Assign Node In Same Y Row as AUV As Priority Node
NO
if One On Same Z Plane As AUV And Other Not
NO
Assign Node In Lower Z Plane As Priority Node
YES
Assign Node In Same Z Plane As AUV As Priority Node

Figure 27: Relative node location heuristics determine priority node among nodes of equal distance from AUV.

and in the same vertical plane as the AUV. Once all of the nodes in the node list have been compared the priority node as indicated by the best node pointer is returned.

F. INTERMEDIATE-LEVEL PATH PLANNING ALGORITHM

When a priority node has been determined the AUV is advanced using an intermediate-level path planning algorithm. The algorithm is initiated whether the priority node is adjacent to the AUV or many nodes distant. The A-Star algorithm was chosen for this implementation of the minefield search because it guarantees an optimal path.

1. A-Star Theme

The A-Star search is an agenda based path planning algorithm which uses a combination of a cost function and an evaluation function to determine an optimal path between a start (AUV's current position) and a goal (priority node). The agenda provides a pool of nodes with which to compare the cost and evaluation values for determining the optimal path. Once an optimal path is determined, the AUV performs its transit along the path to the priority node.

2. Cost and Evaluation Functions and Criteria Value

The cost function is the cumulative distance along the best path from the AUV to the current reference node referred to as the "principle node" and then to the "neighbor nodes" of the principle node. The evaluation function is the Euclidean distance measured from the neighbor nodes directly to the priority node. The criteria value assigned to each neighbor node is the sum of the cost and the evaluation functions. The lower the criteria value the more efficient the path. It is essentially a measure of the historical distance needed to travel to the neighbor node combined with an estimate of the future distance to be traveled to the priority node. The criteria value provides a reasonable measure for comparing one possible path to another. The principle node at any given time represents the algorithm's "best guess" of where the optimal path lies. The algorithm therefore tends to analyze directly toward the priority node and is thus very efficient in most obstacle environments.

3. Nodes Available for Analysis

The nodes available to the A-Star search for path analysis consist only of those with the state VISITED. This means that only nodes known to be attainable are used. It is conceivable that ACTIVE nodes could be used but because these blocks have not been thoroughly searched the path between adjacent ACTIVE nodes could conceivably be obstructed by an object not previously detected. It is also possible to assume that FREE nodes are not obstructed by obstacles and allow them to be used. This would equate to performing path planning in unknown waters, resulting in unacceptable inefficiencies. ACTIVE and FREE nodes are therefore not used.

4. A-Star Data Structures

a. Agenda

The agenda is kept in a linked list which is sorted by criteria value from the lowest to the highest. The principle node returned from the agenda is the highest priority node which is located at the head of the list.

b. Path Map

The path map is an array with the same dimensions as the world array. It is used to keep track of the path from the AUV to the node being analyzed.[12] Each node in the path map contains slots for the coordinates of the previous node in the path, the cumulative cost along the path, the state of the node, and a pointer to the corresponding node in the agenda for easy reference. The states of the nodes in the path map array are different from those maintained in the world array. They include:

- NOTEVAL. This indicates that the node has not been evaluated by the A-Star algorithm. It is the default state for all nodes.

- FRONTIER. The node has been analyzed, its criteria value computed and placed on the agenda for evaluation.

- CHECKED. The node has been evaluated and taken off of the agenda. It also indicates that the path from the AUV to the node is the least cost path possible at the time that the node is annotated CHECKED.

Every FRONTIER or CHECKED node has a unique path in the path map which represents the known shortest path from the AUV to that node.

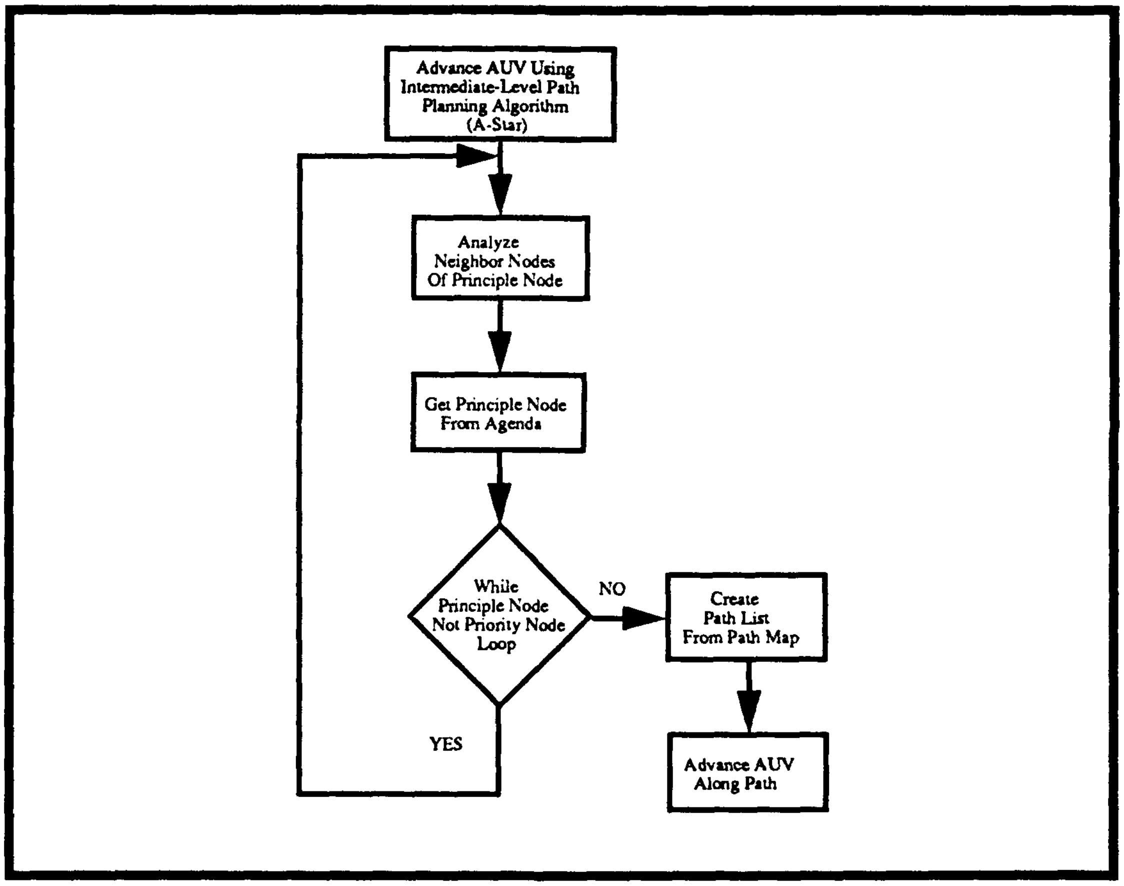
12. In the A-Star search analyzed means that the node's state has been evaluated to determine appropriate follow-on action.

c. *Path List*

This linked list is formed following the A-Star search proper by tracing backward from the priority node to the AUV in the path map. It provides the optimal path for advancing the AUV.

5. A-Star Algorithm

The flow diagram in Figure 28 gives the higher level process for the A-Star intermediate-level path planning algorithm. The principle node is always the CHECKED



Advance AUV Using Intermediate-Level Path Planning Algorithm (A-Star)
Analyze Neighbor Nodes Of Principle Node
Get Principle Node From Agenda
While Principle Node Not Priority Node Loop
NO
YES
Create Path List From Path Map
Advance AUV Along Path

Figure 28: A-Starch Intermediate-Level Path Planning Algorithm

node with the lowest criteria value. Neighbor nodes of the principle node are analyzed and added to the agenda based on their criteria value. A new principle node is then retrieved from the agenda. This cycle repeats until the principle node is the same node as the priority node indicating that the goal has been reached. The path list is then created and the AUV is advanced along this path.

a. Analyzing Nodes Which are Neighbors of the Principle Node

Figure 29 is a flow diagram of the procedures for analyzing the nodes which are neighbors of the principle node.

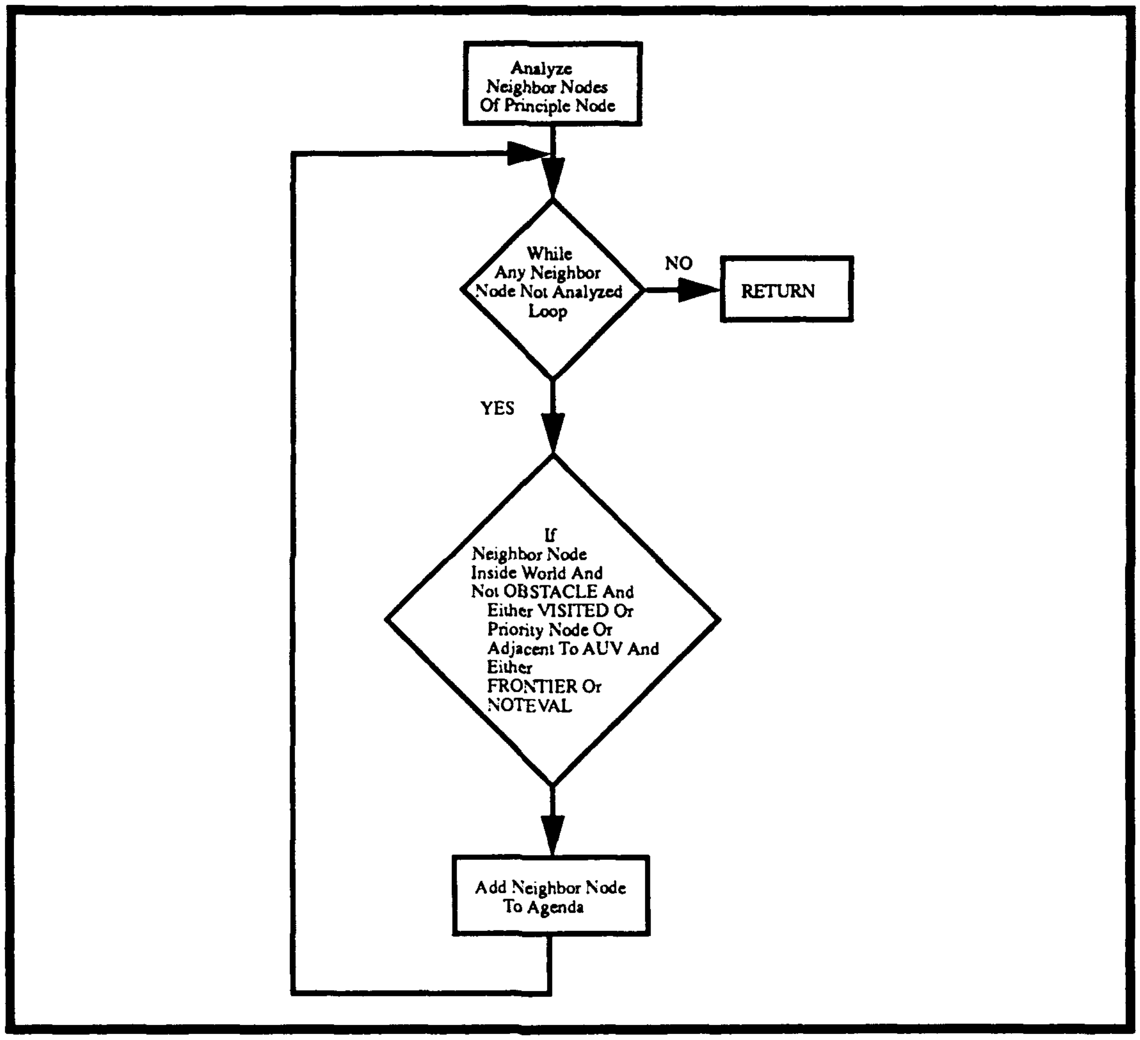

Analyze Neighbor Nodes Of Principle Node
While Any Neighbor Node Not Analyzed Loop
NO
RETURN
YES
If Neighbor Node Inside World And Not OBSTACLE And Either VISITED Or Priority Node Or Adjacent To AUV And Either FRONTIER Or NOTEVAL
Add Neighbor Node To Agenda

Figure 29: Analyzing Nodes Which are Neighbors of the Principle Node

When the A-Star algorithm is initiated, the node where the AUV is located is the principle node. Old principle nodes are continuously replaced by new principle nodes as the search expands. From the principle node, all neighbor nodes are analyzed and their criteria values are computed. If the neighbor node is not an OBSTACLE; is either VISITED, the priority node, or immediately adjacent to the AUV; or if the node's state is either FRONTIER or NOTEVAL then it is added to the agenda.

b. Adding Neighbor Nodes to Agenda

Figure 30 describes the process for adding neighbor nodes to the agenda.If the neighbor node is a FRONTIER node, it has been previously placed on the agenda along

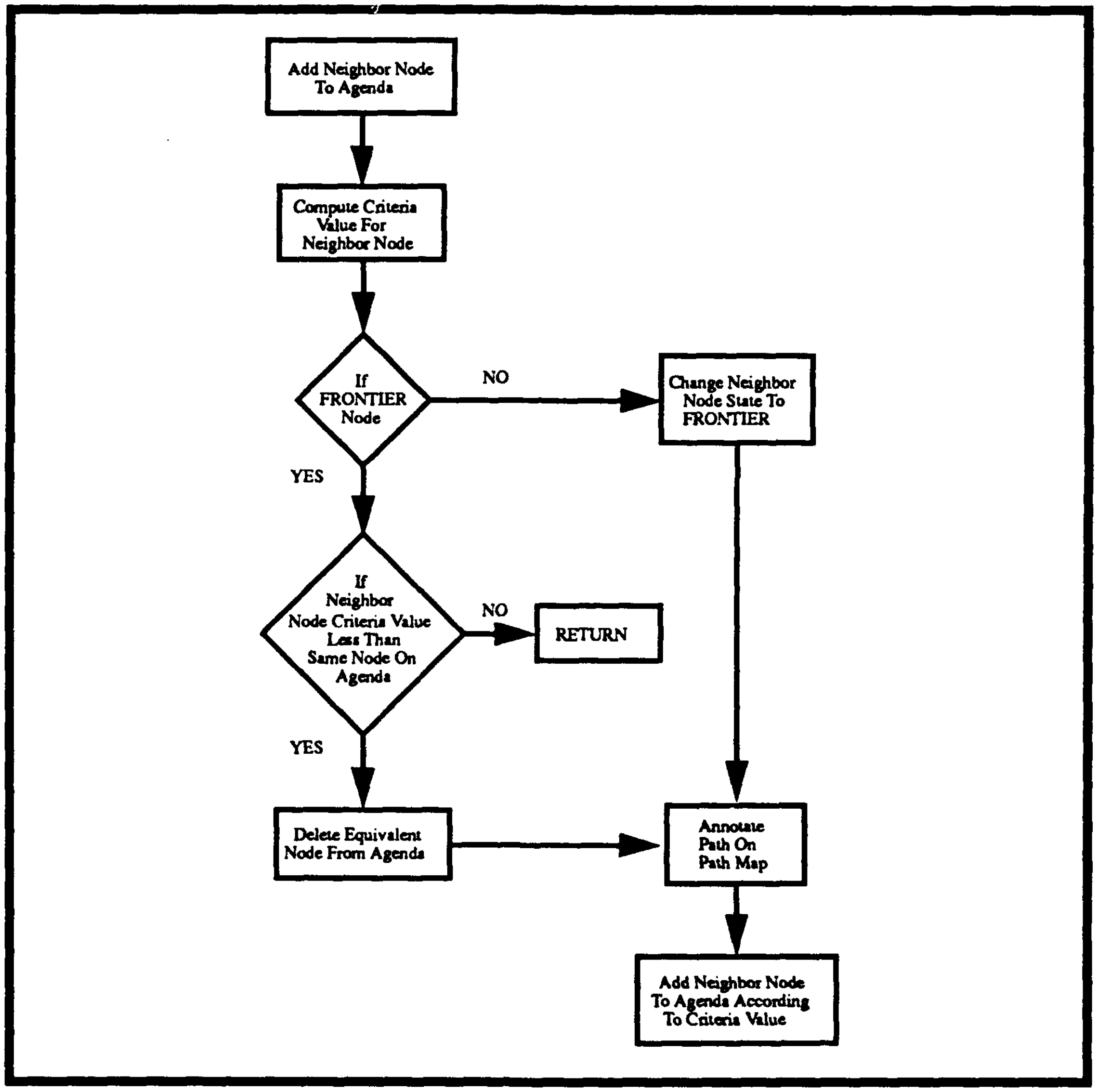

Add Neighbor Node To Agenda
Compute Criteria Value For Neighbor Node
If FRONTIER Node
NO
Change Neighbor Node State To FRONTIER
YES
If Neighbor Node Criteria Value Less Than Same Node On Agenda
NO
RETURN
YES
Delete Equivalent Node From Agenda
Annotate Path On Path Map
Add Neighbor Node To Agenda According To Criteria Value

Figure 30: Adding Nodes to Agenda

with its criteria value when the equivalent neighbor node of another principle node was analyzed. The criteria value of the current neighbor node must be compared to the criteria value of the equivalent node which is already on the agenda. If the criteria value of the current neighbor node is less than the value of the equivalent node on the agenda, the equivalent node is removed from the agenda. The path on the path map is then updated so that the principle node is listed as the predecessor of the current neighbor node. Finally the current neighbor node is added to the agenda according to its criteria value.

If the neighbor node is not a FRONTIER node then it has no equivalent node already on the agenda. The neighbor node's state is changed to FRONTIER, the path on the path map is updated so that the principle node is listed as the predecessor of the neighbor node, and the neighbor node is added to the agenda according to its criteria value.

G. MINEFIELD SEARCH PERFORMANCE

The efficiency ratio, *Eratio*, was devised as a comparative measure of the efficiency with which the AUV searches its environment. *Eratio* is defined as:

$$Eratio = \frac{Distance_{Reference}}{Distance_{Traveled}}.$$

$Distance_{Reference}$ is a function of the minimum distance required to search a minefield void of obstacles corrected for the estimated distance to traverse through obstacles found in this particular minefield.

$$Distance_{Reference} = Nodes_{Total} - (2 \times Obstacles_{Total}).$$

$Distance_{Traveled}$ is a measurement of the actual distance traversed by the AUV from the start to mission completion.

Empirical analysis shows that the *Eratio* varies directly with the complexity of the obstacles in the minefield. Highly complex obstacles such as spiral mazes which overlap numerous sub-areas have the greatest detrimental effect on efficiency. Less complex obstacles which are indicative of those found in the real ocean environment have a very minor effect on efficiency. *Eratios* varried from 1.0 for an obstacle free environment to 0.04 for a highly complex obstacle field.

H. EXPERIMENTAL RESULTS

1. Minefield Search Algorithm Verification

The minefield search algorithm has been verified against over 24 test routines which exhaustively represent the mission requirements for static obstacle fields established earlier in this chapter. The test routines included a variety of obstacle templates of various densities and complexities.

a. Search All Attainable Blocks

In each test case the AUV visited every attainable node, searching the corresponding blocks.

b. Start Search from any Node Not Obstructed by an Obstacle

The AUV was placed in FREE nodes throughout the minefield search areas to begin its missions. In all cases, the AUV successfully completed its mission. When placed in the interior of a solid OBSTACLE the vehicle immediately terminated its mission.

c. Ignore Nodes Which are Unattainable

The AUV did not attempt to evaluate any unattainable nodes. This included nodes located in the interior of obstacles, above the ocean surface and within the ocean bottom.

d. *Perform Coarse Maneuvering Around Obstacles*

In all test scenarios, the AUV successfully maneuvered around all obstacles including those representing the ocean bottom and the ocean surface. It is significant to note that this algorithm effectively serves as a coarse terrain following algorithm.

e. *Successfully Search in Presence of Stove Pipe and Blind Alley Obstacle Anomalies*

While many search algorithms tend to get stuck in local minima when confronted with stove pipes and blind alleys, the minefield search algorithm successfully handled these pervasive problems. Variations on these obstacles included nearly all possible orientations in both two and three dimensions. In all cases, the AUV searched the interior of these obstacles when there was an attainable path and ignored the interior when a path was not available.

2. Ladder Search In Prioritized and Non-Prioritized Sub-Area Environment

Tests were conducted which compared the efficiency of the minefield search based on prioritized sub-areas to one based simply on the ladder search theme. Regardless of the density of the obstacle field, the non-prioritized sub-area search returned an *Eratio* which was approximately four percent more efficient than the prioritized sub-area search provided. However, the non-prioritized sub-area search normally left many gaps in its coverage which were generally not filled in until late in the search mission (again see Figure 17).

I. CHANGING OBSTACLE ENVIRONMENT CONSIDERATIONS

The minefield search algorithm not only searches around stationary obstacles but also adjusts for most moving obstacle situations. As the AUV advances along the path returned in the path list it checks for an obstacle at each node to which it is advancing. If a new obstacle is detected at that node, the AUV terminates advancing, updates the node to reflect OBSTACLE and deletes the path list. The algorithm then reinitiates intermediate-level path planning to the priority node using the AUV's current position as the starting point and the revised world map of VISITED nodes.

While this adjustment works for most conceivable changes in the obstacle environment there are three cases where it fails:

- The obstacle environment has changed so that the AUV is completely surrounded by obstacles and no possible path is available to the priority node.
- The obstacle environment has changed so that the priority node is completely surrounded by obstacles.
- The obstacle environment is changing so dynamically that the AUV perceives that it or the priority node are completely surrounded even though this may not be the case.

J. CONCLUSIONS AND FUTURE ALGORITHM DEVELOPMENT

The minefield search algorithm presented here is not an optimal search solution. It does, however, successfully meet all of the mission requirements established earlier in this chapter. What the algorithm lacks in search optimality it makes up for in adherence to the important philosophy of thoroughly and systematically searching large sections of ocean. It is extremely flexible. It does not require an *a priori* world map. It can be implemented in any conceivable minefield search environment. The only necessary mission preparation is to define the search area boundaries and the search track spacing. While the three-dimensional algorithm provides the most flexibility, the two-dimensional algorithm may be used in either two or two-and-one-half-dimensional search scenarios.

Future algorithm development should first focus on ways to optimize the search solution while maintaining the prioritized sub-area philosophy. An immediate improvement would be to substitute a best first search for the Euclidean distance method for determining the priority node in the priority sub-area. This would eliminate the efficiency problem presented when the closest node based on Euclidean distance is not the closest node based on the actual path required to get to the node.

Another immediate focus should be the dynamic interface between a graphic simulator and the search algorithm so that experiments with moving objects can be conducted.

Finally, alternatives to the prioritized sub-area method also need further investigation. These alternatives could include specialized heuristic methods which utilize artificial intelligence techniques.

V. MINEFIELD SEARCH SIMULATION

A. EVALUATION THROUGH VISUALIZATION

The testing of NPS AUV software is a difficult task because of the remote operating nature of the AUV, the limited number of pool test missions available and the limited types of mission that can be tested in the pool environment. These limitations can be partly overcome by testing software performance in computer simulations. Such testing provides important software performance validation prior to committing to actual AUV pool, or eventually open ocean, test missions.[Ref. 14]

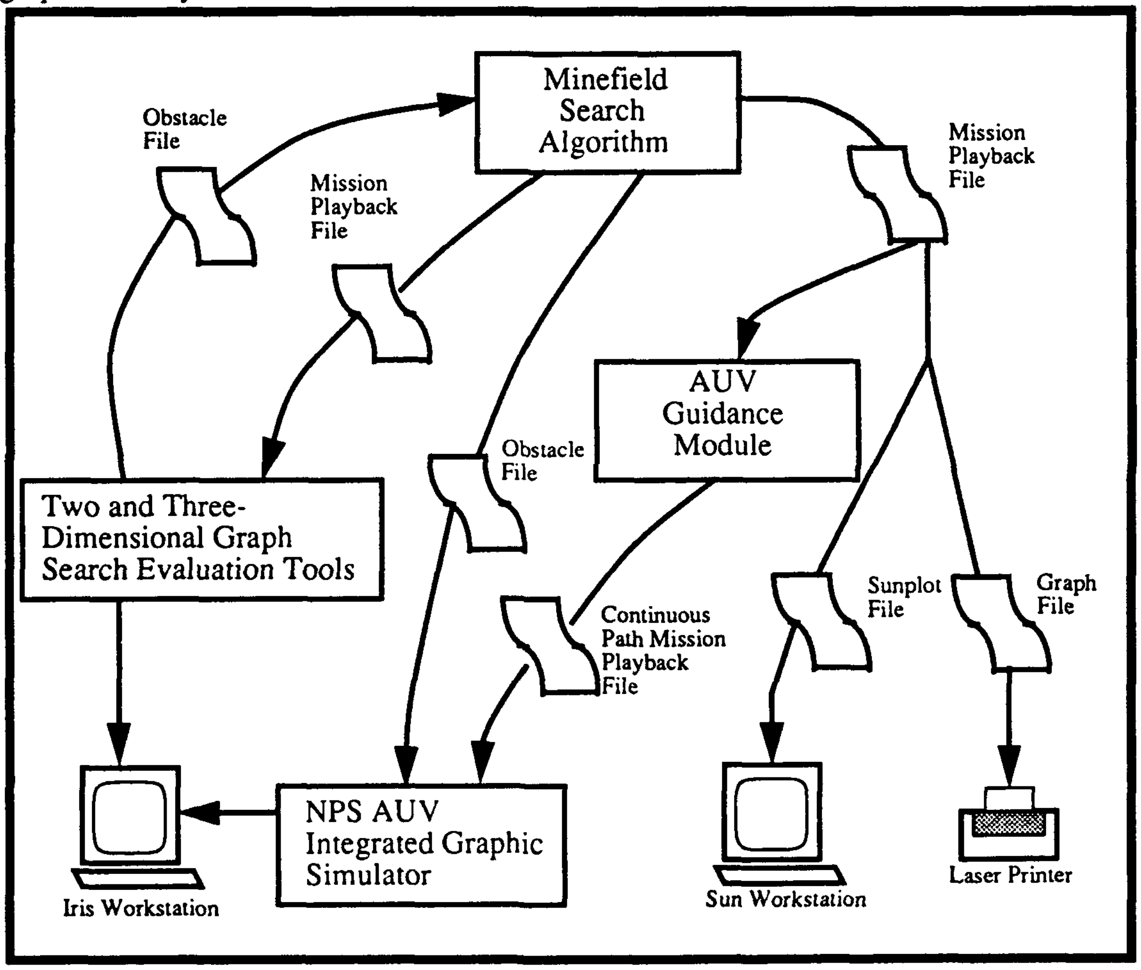
There are several levels of sophistication for evaluating the results of software testing. One of the most common but least sophisticated methods is to analyze the output of raw data. This method is slow, tedious, and prone to mistakes. In the case of the NPS AUV there are 17 floating point numbers output to file every one-tenth of a second. The amount of data to be analyzed in even very short missions becomes overwhelming. Because humans are "visual beings" they have great difficulty assimilating these large amounts of raw data and understanding the results.[Ref. 28]

The next level of sophistication is the output of data in the form of static two-dimensional graphs. While this method allows close analysis of two-dimensional problems, it becomes very confusing when analyzing complex problems in higher dimensions.

Next is the two and two-and-one-half-dimensional dynamic analysis of both discrete and continuous data. This type of analysis is highly effective for analyzing information which changes over time. However, like the two-dimensional graph it is difficult to understand when analyzing problems of higher dimensions. Both the Two and Three-Dimensional Graph Search Evaluation Tools are used for this type of analysis of the minefield search algorithm.

Finally there is the dynamic analysis of data using three-dimensional graphical output. This method allows flexible and effective analysis of data in up to three physical dimensions plus the temporal dimension. Because the user can change his perspective, problems which may have been overlooked in less sophisticated analysis methods often become apparent. The NPS AUV Integrated Graphic Simulator utilizes this method to visualize AUV software performance. One aspect of this simulator which has not yet come to fruition is the incorporation of a vehicle guidance module.

Figure 31 describes the flow of information for performing numerous types of graphical analysis of AUV minefield search missions.



Minefield Search Algorithm
Obstacle File
Mission Playback File
Mission Playback File
AUV Guidance Module
Obstacle File
Two and Three-Dimensional Graph Search Evaluation Tools
Continuous Path Mission Playback File
Sunplot File
Graph File
Iris Workstation
NPS AUV Integrated Graphic Simulator
Sun Workstation
Laser Printer

Figure 31: Minefield Search Information Flow Diagram For Graphical Analysis

1. Sunview "graph" and "sunplot" Two-Dimensional Analysis

A static two-dimensional image of vehicle track can be displayed by using the Sunview "graph" command offered on UNIX.[1] This image may be sent to a printer or can be displayed at the Sun Workstation terminal using the "sunplot" command.[2] Figure 32 shows the vehicle's track during a three-dimensional minefield search mission using the graph command. While evaluation of mission performance can be performed more easily using this method than by reading raw data output it is still difficult to analyze in three-dimensional complex environments.

2. Two-Dimensional Graph Search Evaluation Tool

The Two-Dimensional Graph Search Evaluation Tool is implemented on a Silicon Graphics Iris Workstation.[3] A graphic display of this tool's user interface is provided in Figure 33. The following features make this tool highly useful when evaluating two-dimensional discrete graph problems:

- Input is received from a file and consists of state, x and y values for each action to be displayed.

- Output is the color graphic display showing changes in the state of each node as they occur.

- Upload and playback of minefield search missions are performed through the mouse menu.

- Multiple playback speeds may be selected through the mouse menu.

- Mission playback may be paused and resumed through the mouse menu.

- The state of individual nodes may be changed to OBSTACLE or FREE using the point and click feature of the mouse. Multiple obstacle fields may be constructed in this manner.

- An obstacle field may be saved to a file through the mouse menu for use by the

1. UNIX is a registered trademark of AT&T Bell Laboratories.
2. Sun and Sunview are trademarks of Sun Microsystems, Inc.
3. IRIS is a trademark of Silicon Graphics, Inc.

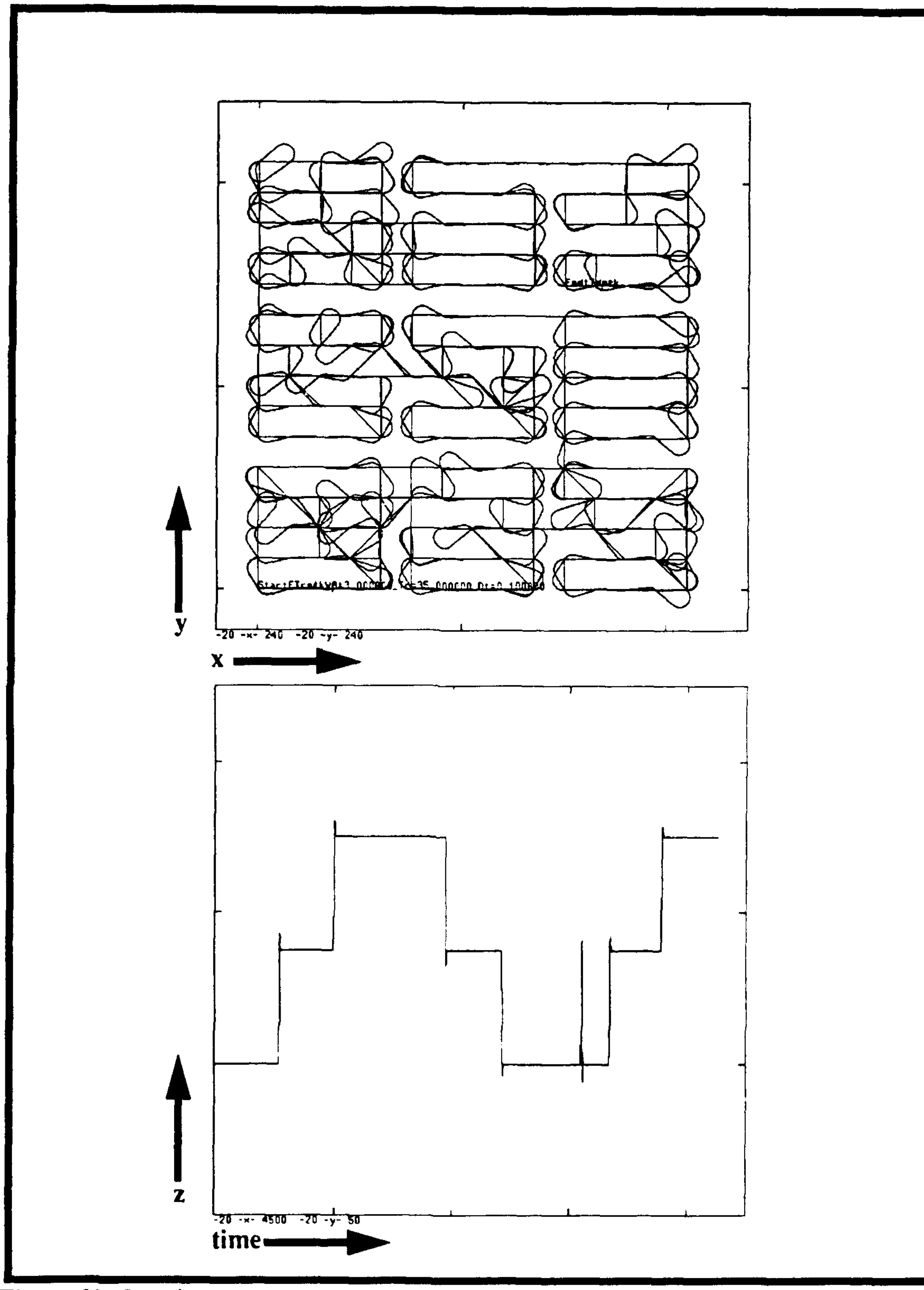


Figure 32: Sunview "graph" Display of Three-Dimensional Minefield Search Mission

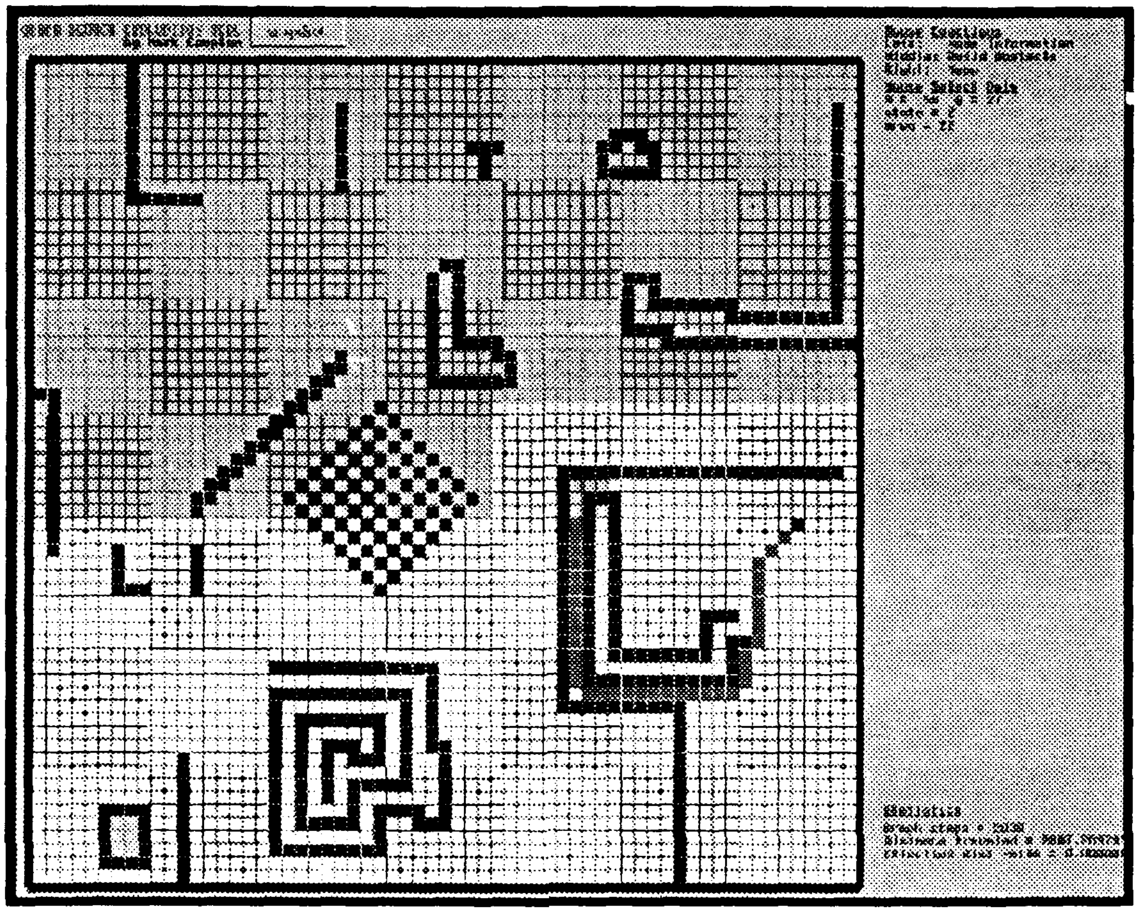


Figure 33: Two-Dimensional Graph Search Evaluation Tool Display

minefield search algorithm.

- Position, sub-area number and state may be displayed for each node with the point and click of the mouse.

- Distance traveled, the number of nodes traversed and efficiency ratio are displayed.

- Search area dimensions may be changed rapidly through global variables.

3. Three-Dimensional Graph Search Evaluation Tool

This tool, depicted in Figure 34, provides a visual display of the change in node states on each level of a three dimensional search, one level at a time. The features are

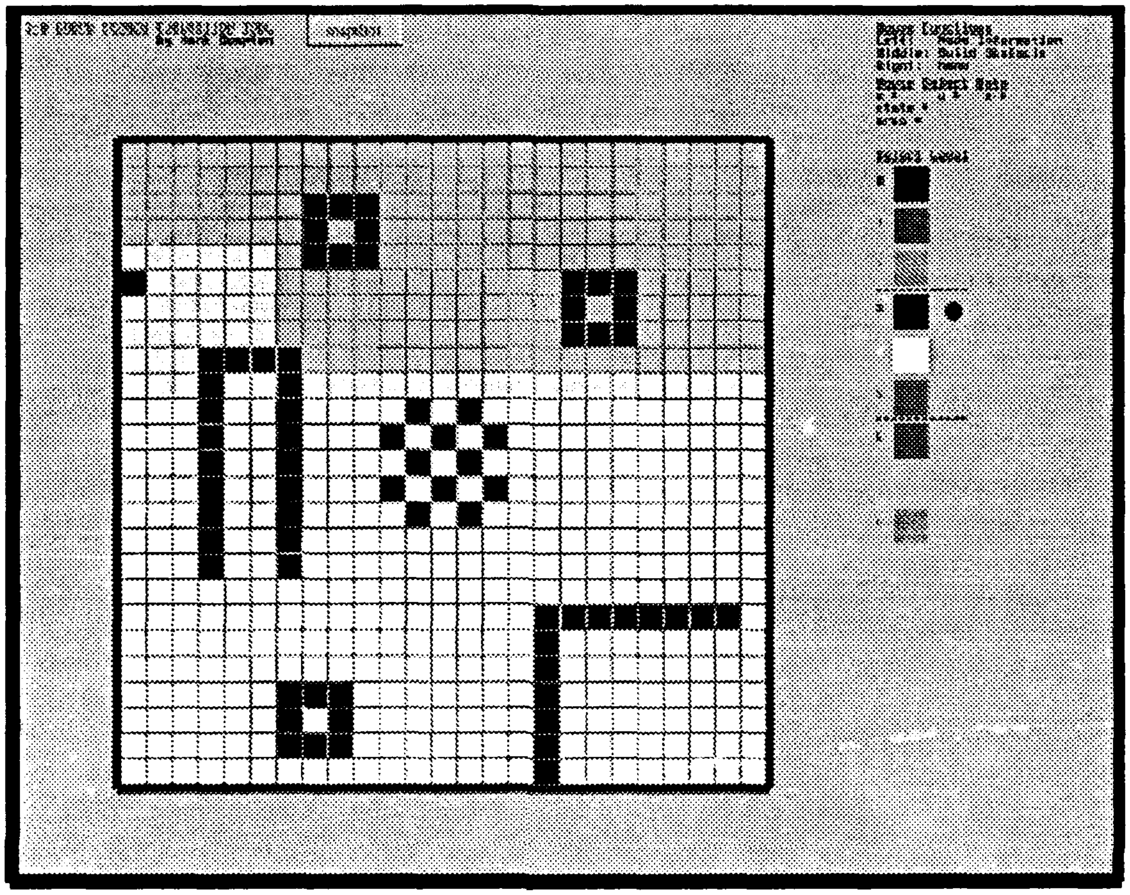


Figure 34: Three-Dimensional Graph Search Evaluation Tool

identical to those of the Two-Dimensional Graph Search Evaluation Tool with the following additions or exceptions:

- Inputs include a z position component.

- Levels may be viewed individually through a mouse point and click on the select level panel to the right of the search area.

- Different levels are represented by different colors, as well as by a pointer to the selected level on the select level panel, for easy identification.

- Three-dimensional obstacle fields may be created and stored to a file.

- The displayed level corresponds to the latest change in the state of a node on that level, allowing the user to visualize significant mission events.

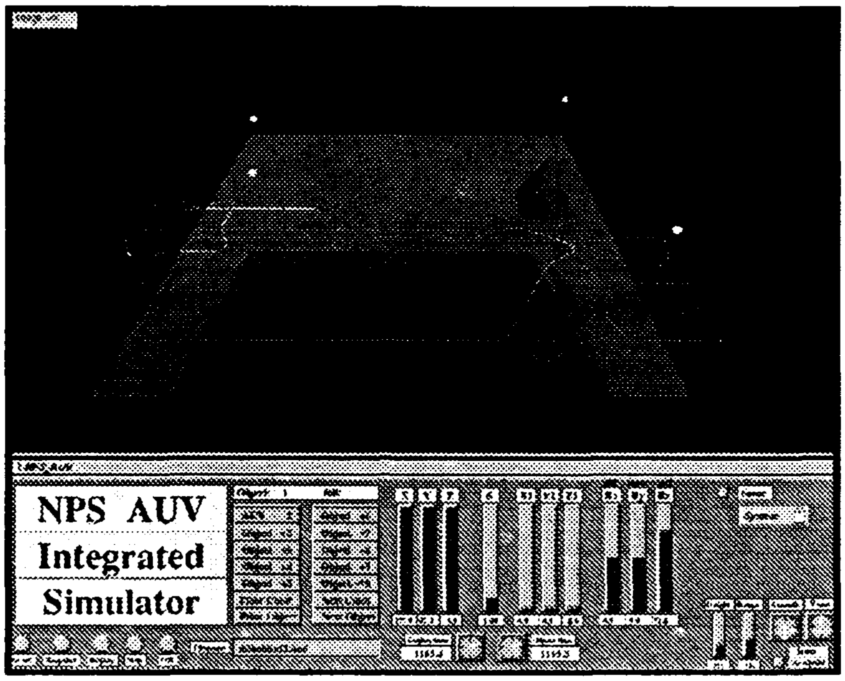


Figure 35: NPS Integrated Graphic Simulator Playback of Minefield Search

B. THREE-DIMENSIONAL LINE-OF-SIGHT GUIDANCE MODULE

This section presents a simple Euler based kinematics line-of-sight guidance model which takes waypoint inputs from the minefield search algorithm, computes vehicle track, and outputs continuous postures to the simulator.

There are many integration methods for deriving positional information from the acceleration vector. T. Jurewicz proposes a modified Euler method which utilizes an average predicted velocity [Ref. 11]. C. Magrino in conjunction with Y. Kanayama present a method which utilizes spatial posture curvature values to determine vehicle path and reference and error postures to return the vehicle to predefined paths [Ref. 13].

While these method have many advantages they may be unnecessarily complex for the purpose of high level path analysis in minefield search playback. The following model was chosen for this particular implementation.

e. Vehicle Posture Updates

Vehicle postures are updated as a function of the change in time Δt, vehicle velocity v, elevation direction of change ε, elevation rate e, azimuth direction of change α and azimuth rate a. The posture of the vehicle at $t + 1$ is equal to:

$$P_{t+1} = (x_{t+1}, y_{t+1}, z_{t+1}, \phi_{t+1}, \theta_{t+1}, \psi_{t+1})$$

where

$$x_{t+1} = x_t \pm ((v \times \Delta t) \times \cos\psi),$$

$$y_{t+1} = y_t \pm ((v \times \Delta t) \times \sin\psi),$$

$$z_{t+1} = z_t \pm ((v \times \Delta t) \times \sin\theta),$$

$$\theta_{t+1} = \theta_t + (\varepsilon \times (e \times \Delta t)), \text{and}$$

$$\psi_{t+1} = \psi_t + (\alpha \times (a \times \Delta t)).$$

f. Relative Orientation

To determined guidance commands necessary to achieve a desired waypoint the orientation of the vehicle at a given time relative to the waypoint W_n must be determined. This orientation is expressed as two components: relative elevation Θ and relative azimuth Ψ. Relative elevation at time t is computed as:

$$\Theta_t = \operatorname{atan}\frac{z_t - z_n}{\sqrt{(x_t - x_n)^2 + (y_t - y_n)^2}}.$$

Relative azimuth at time t is computed as:

$$\Psi_t = \operatorname{atan}\frac{y_t - y_n}{x_t - x_n}.$$

g. *Line-of-Sight Vehicle Guidance*

Relative orientations Θ and Ψ are compared to the elevation and azimuth components θ and ψ of the vehicle posture to determine elevation direction of change ε and azimuth direction of change α commands for line-of-sight guidance. Both direction of change commands are applied repeatedly, at each Δt, in the shortest angular direction in an attempt to achieve a zero relative orientation.

2. Waypoint Sequencing

The NPS AUV guidance system sequences from the current waypoint to the next when the vehicle is within a predefined radius of the current waypoint [Ref. 6]. The close proximity of sequential waypoints to each other combined with a limited AUV turning radius can cause a "spiraling effect" where the vehicle continuously tries but is unable to capture a waypoint.

The waypoint sequencing method presented here avoids the spiraling effect by defining an imaginary boundary perpendicular to the *x,y* plane which lies perpendicular to the orientation azimuth component A (e.g. $A \pm \frac{\pi}{2}$) of the incoming track to each waypoint W_n from its predecessor W_{n-1}. The exception is the first waypoint W_1 where the boundary is perpendicular to the orientation azimuth component of the outgoing track W_1 to W_2 (see Figure 36). Elevation components of orientation with respect to the vehicle and the waypoints are ignored (waypoint sequencing analysis uses only *x,y* planar elements). This decision has a precedence in that aircraft inertial navigation units typically ignore the vertical reference when sequencing from one waypoint to the next.

The algorithm sequences through each waypoint and its successor comparing the vehicle's relative orientation azimuth component Ψ_{W_n} to the boundaries which point perpendicular to the incoming track orientation azimuth component $A_{n-1,n} \pm \frac{\pi}{2}$ of each waypoint. When Ψ_{W_n} is found to lie within the relatively forward hemisphere of W_{n+1} and the aft hemisphere of W_n, the vehicle is determined to between the two waypoints. If W_n was previously the destination waypoint then W_{n+1} becomes the new destination waypoint. Figure 36 depicts this process.

The waypoint sequencing algorithm is implemented each time the vehicle updates its posture until the final waypoint is reached and the mission terminates.

C. CONCLUSIONS AND FUTURE DEVELOPMENT

The visualization tools presented here have been invaluable in performing minefield search algorithm development and evaluation. Each level of sophistication in visualization had its own particular advantage in analyzing different aspects of the mission. The Graph Search Evaluation Tools were particularly useful in performing discrete analysis of the AUV's chosen paths. The NPS AUV Integrated Graphic Simulator, on the other hand, was invaluable for viewing the continuous path of the AUV in three dimensions.

Another important factor was the design choice to completely separate the minefield search algorithm from the simulators. This permitted access to the simulators for mission analysis even when the minefield search code contained flaws. It proved critical in determining the source of many of those flaws.

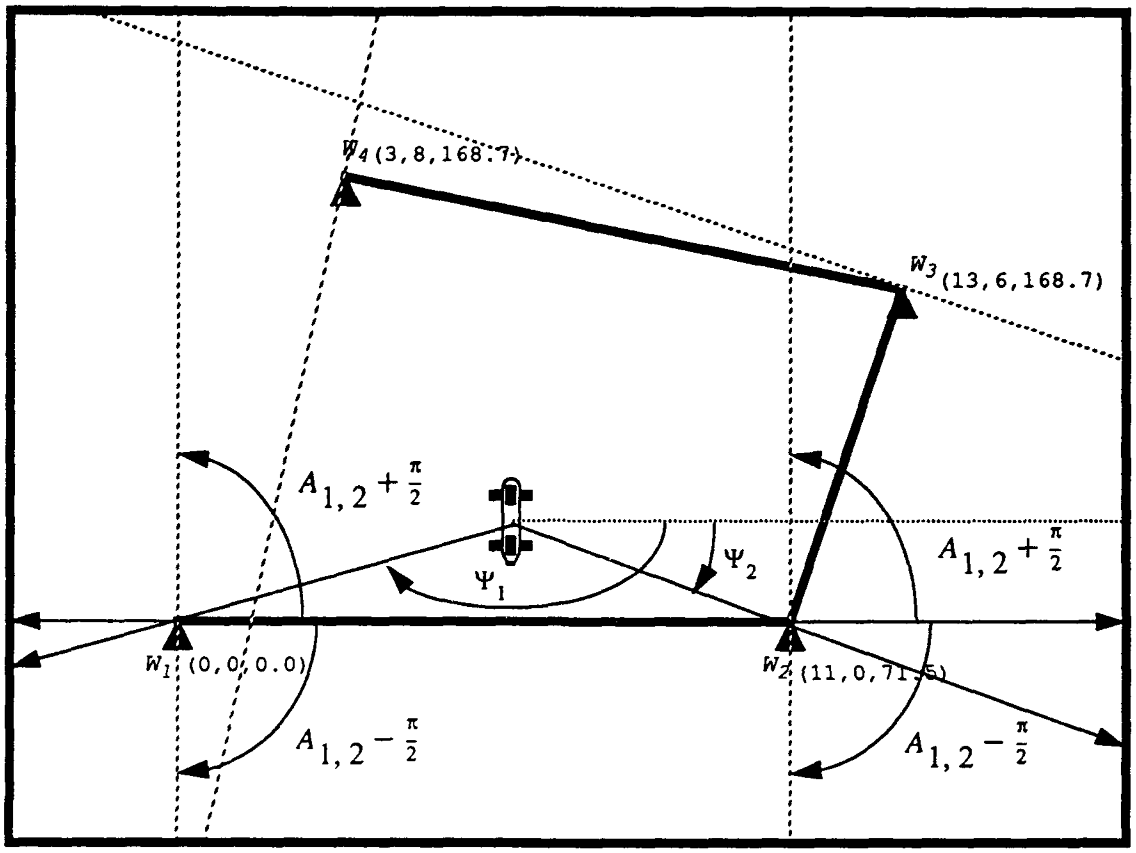


Figure 36: Waypoint sequencing determination based on azimuth orientation component of AUV to orientation of successive waypoints.

A future development should be the testing of the minefield search algorithm within the Gespac component of the NPS AUV Integrated Graphic Simulator once the guidance module and other software modules have been loaded and evaluated. This testing will ensure that the software is directly portable to the actual NPS AUV. It will also eliminate the need for the add-on guidance package presented in this chapter.

VI. AUTONOMOUS SONAR CLASSIFICATION USING EXPERT SYSTEMS

A. INTRODUCTION

Intelligent vehicles will play a major role in future underwater missions such as the minefield search mission presented in this thesis. A critical requirement for independent behavior by such vehicles is autonomous analysis of complex and variable ocean environments. This is a notoriously difficult task, even when human operators use sophisticated sensors and powerful processors.

Although much work has been done in vision processing for mobile robots, additional research has been needed on interpretation of observed scenes and terrain [Ref. 29]. Numerous approaches to the general object-recognition problem are presented in [Ref. 30]. Both of these references can be found in [Ref. 31], an essential collection of surveys, tutorials and fundamental research papers regarding mobile robot sensor perception, mapping and navigation. Other references included in [Ref. 31] are [Ref. 32] and [Ref. 33].

Independent and meaningful interpretation of sensor data is a principal prerequisite for accomplishing high-level AUV missions and behaviors. A number of universities and laboratories are conducting autonomous underwater vehicle (AUV) research and development that involves a wide variety of sensor types and sensor interpretation methods. The Defense Advanced Research Projects Agency (DARPA) Unmanned Undersea Vehicle (UUV) uses sidescan sonar and neural network classification for underwater mine detection [Ref. 15]. Woods Hole Oceanographic Institution has used sidescan sonar, stochastic backprojection and a variety of vision processing techniques and sea floor shape information to create three-dimensional bottom images [Ref. 34]. The University of New Hampshire Experimental Autonomous Vehicle (EAVE) III uses depth profiling, acoustic long baseline navigation and comparison with a world model to detect bottom objects [Ref. 35]. Numerous other examples of sensor data interpretation exist. In contrast to most methods, the sonar classification system presented in this chapter uses parametric regression, geometric analysis and expert system heuristics to create classifiable object types. An advantage of this method is that progressively higher levels of object abstraction are possible.

B. OVERVIEW

The objective of this chapter is to present a method for autonomous classification of underwater objects. This is achieved using geometric sonar analysis techniques and an expert system for heuristic reasoning.

This research effectively demonstrates that geometric analysis can be combined with an expert system to process, analyze and classify active sonar range and bearing data in support of AUV operations. Figure 37 shows how low-level sonar data is processed to produce increasingly complex geometric objects and high-level classification outputs.

Geometric analysis can distill large amounts of sonar data into useful information which can be used to make logical and informed decisions. The primary difficulty in geometric sonar analysis is that active sonar signal returns are inherently noisy and unconnected. Parametric regression is a robust method of least-squares line fitting that permits precise geometric analysis of range and bearing data.[Ref. 36] Generated regression lines are provided to a polyhedron-building algorithm to create geometric objects. Geometric object attributes can then be compared to known object types through the rule-based pattern-matching capabilities of an expert system, resulting in object classification.

The possible types of object classes to be detected are typically limited in number and somewhat predictable given *a priori* knowledge of the underwater environment. Geographic objects to be detected and classified include the ocean bottom, sea mountains, valleys, rock outcroppings and walls. Biological objects include fish, kelp, scuba divers and large animals such as dolphin or whales. Man-made objects include ships, submarines, torpedoes, mines, nets, pipes and cables. These object classes of interest are listed in Table DThe relatively small number of underwater objects of interest simplifies sonar

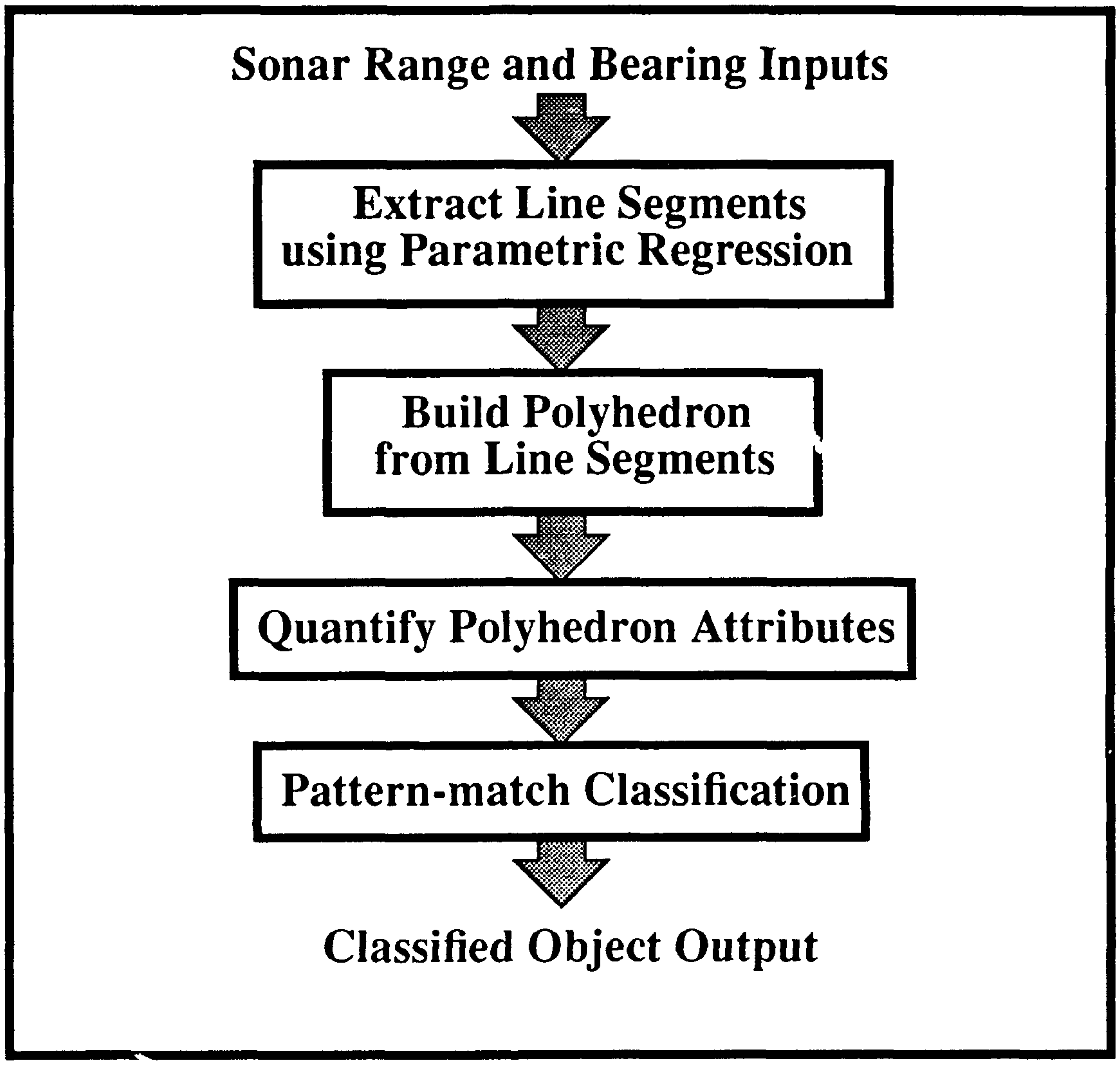

Sonar Range and Bearing Inputs
Extract Line Segments using Parametric Regression
Build Polyhedron from Line Segments
Quantify Polyhedron Attributes
Pattern-match Classification
Classified Object Output

Figure 37: Autonomous Sonar Classification Process Diagram

classification criteria. Primary expert system outputs are location, size, and classification of all sonar contacts.

Expert systems are an established methodology that can effectively and clearly represent specialized human knowledge using algorithms and heuristic rules. Typically the functions performed by an expert system would otherwise require human action by a knowledgeable expert. The expert system approach is applicable to a wide variety of complex problems, even when no single expert understands all aspects of a particular problem domain.

TABLE D: EXAMPLE UNDERWATER OBJECT CLASSIFICATION TYPES

Geographic	Biological	Man-made
Ocean Bottom	Fish	Ship
Sea Mountain	Kelp	Submarine
Valley	Scuba Diver	Mine
Rock Outcropping	Dolphin	Torpedo
Wall	Whale	Net
Sea Surface	Shark	Pipe or Cable
Unknown		

The use of real world data is important for development and verification of a sonar classification expert system. Naval Postgraduate School (NPS) students and faculty have designed and built a working AUV which can be used to provide a variety of classifiable sonar data. Successful examples of expert system classifications using NPS AUV sonar data are described in detail.

The expert system approach also appears to be usable for sensor fusion using a wide variety of sonar types as well as non-acoustic sensors such as laser rangefinders and video. Many exciting future applications should be possible using expert system methods.

C. GEOMETRIC ANALYSIS OF SONAR DATA

1. General Characteristics of Active Sonar Data

Outputs common to practically all active sonars are range and bearing from the sonar transducer to a contact, if any is detected. Posture of an underwater vehicle includes a three-dimensional position coordinate, as well as vehicle attitude consisting of roll, elevation and azimuth orientations. The relative position of each sonar return is combined with vehicle posture using vector addition to yield a precise three-dimensional coordinate.

In this chapter the term "sonar data" refers to simultaneous sonar range and bearing data returned from an active sonar transmission.

2. Geometric Primitives and Object Attribute Definitions

Sonar data can be analyzed to produce geometric forms such as points, lines or polyhedra. Precise definitions of geometric primitives and object attributes are necessary for predictable and repeatable sonar classifier performance. It is important that the theoretical basis of a sonar classification expert system be both mathematically rigorous and as general as possible in order to allow increasingly sophisticated analysis of data. A formal geometry-based approach also permits expert system compatibility with a wide variety of sonar types.

The geometric primitives considered by this expert system are point, line segment, polyhedron and cylindrical polyhedron (i.e. a three-dimensional polyhedron that extends vertically up and down from a planar polygon perimeter). Object attributes include centroid position, depth, length, width, height, perimeter, cross-sectional area, thinness, and volume. Indirect attributes such as positional accuracy, confidence factor, inferred edges and hidden edges are also evaluated.

Additional geometric primitives and object attributes can be defined as necessary to utilize the more sophisticated data available from sector scanning, two-dimensional swath or three-dimensional multi-beam sonars. Similar approaches using curved shapes such as circles, ellipses or conics would also be compatible.[Ref. 33]

3. Extracting Line Segments using Parametric Regression

Linear relationships described by sets of discrete data are typically found using standard linear regression analysis, commonly known as least-squares fit. This method is widely used but has a significant limitation in that regression calculations on (x,y) coordinate points parallel to the y-axis result in divide-by-zero singularities for slope and mathematically undefined regression results. Since typical unconstrained sonar data may lie along any three-dimensional orientation, a different method is needed for autonomous fitting of best-approximation line segments to a series of discrete sonar returns.

The parametric regression method utilizes a polar coordinate derivation of linear regression analysis to provide a robust and accurate least-squares fit of line segments to sequences of data points. This method has been developed in detail and is particularly well suited for geometric analysis of real-world sonar data. [Ref. 37] [Ref. 38] [Ref. 36] [Ref. 6] Associated with each regression line segment is an elliptical thinness term which can be used as a metric for line segment accuracy and data variance. Figure 38 shows a typical parametric regression line segment fit to a set of sonar returns.

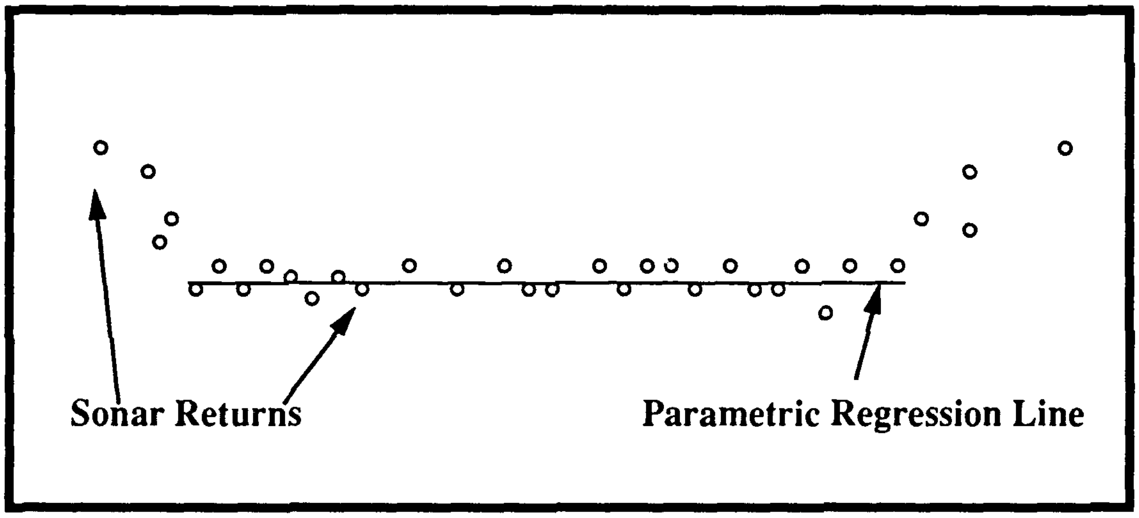


Figure 38: Typical Parametric Regression Line Fit

A further significant benefit of parametric regression analysis is that it is a sequential algorithm which provides immediate incremental improvements upon receipt of each individual data point. The sequential nature of this algorithm makes it highly suited for real-time operations which must meet immediate response requirements. Real-time vehicles cannot afford to wait for intermittently time-consuming sonar analysis when excessive delay might jeopardize navigational safety.

4. Building a Polyhedron from Line Segments

Parametric regression provides linear one-dimensional geometric primitives. However line segments by themselves are insufficient for thorough two-dimensional spatial reasoning or object classification. A polyhedron-building algorithm is presented here as a means of constructing two-dimensional geometric objects from a sequence of regression line segments. In this context the polyhedron-building algorithm is a logical extension to the parametric regression algorithm.

One important assumption used when building polyhedra is that underwater contacts of interest have predominantly convex shapes, i.e. they contain no large concave depressions or cavities. This assumption permits clear delineation of independent object boundaries. Analysis of an actual concave object results in the definition of adjacent convex objects. Higher-level analysis at the heuristic level can be used to clump adjacent objects if needed.

Note that the orientation of vehicle sonar relative to detected objects is a critical consideration in the polyhedron building algorithm, since spatial relationships are equally dependent on sensor perspective and actual object shape.

Polyhedron building begins with a single line segment produced by parametric regression analysis of continuous sonar data. Each following segment from regression analysis on the same sensor is compared to the previous segment. If the follow-on segment meets proximity and orientation criteria, then it is considered to be another part of the same geometric object. This segment comparison process is repeated until proximity or orientation criteria fail, at which time the previous geometric object is complete and the follow-on segment becomes the beginning segment of a new geometric object.

Proximity is measured between the end point of the most recently correlated line segment and the start point of the next segment to be considered. The proximity criterion is typically small and restrictive (e.g. less than 1 foot) in order to permit discrimination between adjacent objects. The proximity criterion must be met prior to comparing relative orientation for geometric object extension.

Orientation comparisons are made to determine whether adjacent segments are colinear, convex or concave. The colinear test allows a reasonable error bound (e.g. 10) in order to account for sonar noise and line-fitting approximations. Colinear segments are acceptable for geometric object extension (Figure 39).

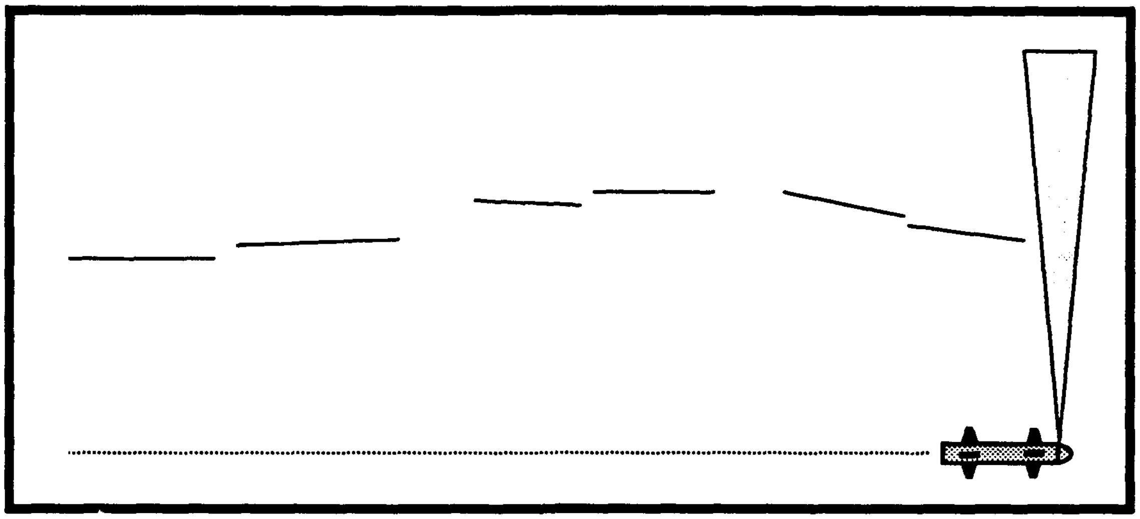


Figure 39: Examples of Colinear Regression Line Segments

The convex test measures whether the follow-on segment direction points farther away from the sensor's perspective than the previous segment. Convex segments are also acceptable for geometric object extension (Figure 40).

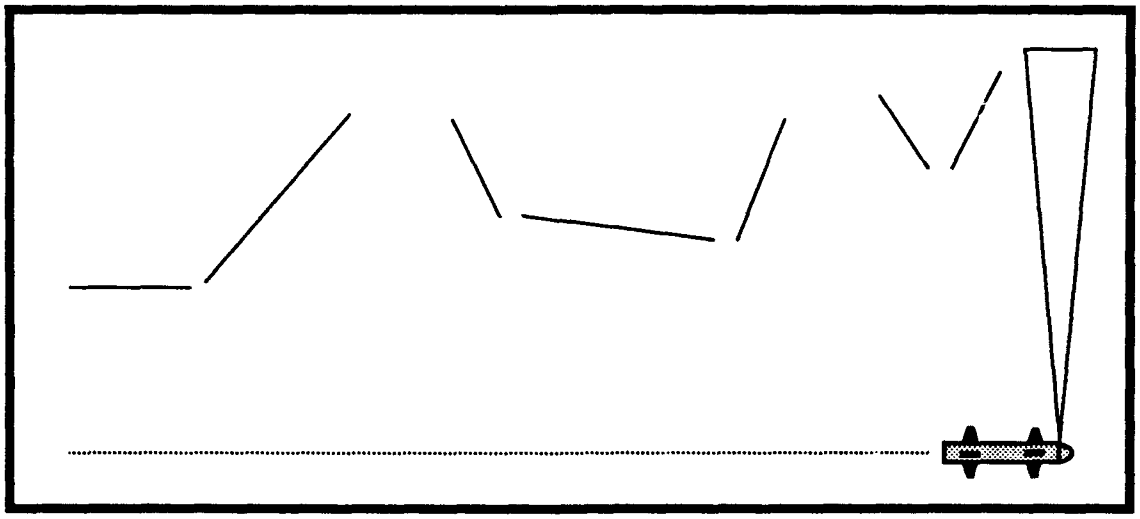


Figure 40: Examples of Convex Regression Line Segments

The concave test measures whether the follow-on segment direction points closer towards the sensor's perspective than the previous segment, in effect defining the boundaries of a hole. Concave line segment relative orientations indicate a break between separate convex geometric objects (Figure 41). The follow-on segment is used to start a new polyhedron.

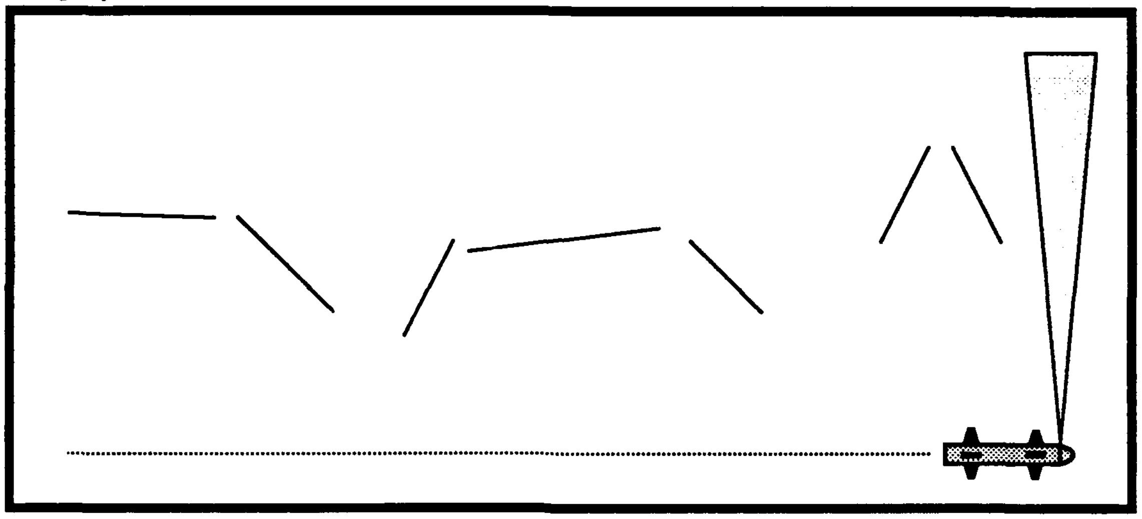


Figure 41: Examples of Concave Regression Line Segments

Inferred edges are presumed to exist between each pair of the sequential detected edges that make up a polyhedron. A single hidden edge is presumed to exist between the start point and end point of a particular object. The classifier should recognize, however, that such hidden edges may be completely inaccurate since the actual hidden sides of the object were obscured from the sonar.

In summary, the polyhedron-building algorithm correlates regression line segments into two-dimensional polyhedral objects. This method enables the application of computational geometry techniques to analyze large volumes of discrete range and bearing data. Figure 42 illustrates the polyhedron-building algorithm.

5. Quantifying Polyhedron Attributes

The attributes which are used to classify objects should be precisely defined and calculated, wherever possible. For example, attributes such as depth, length, width and height are directly measurable using calculated sonar positions. Object perimeter can be determined by first summing the lengths of all correlated line segments, and then adding the lengths of all inferred and hidden edges that are presumed to exist between detected

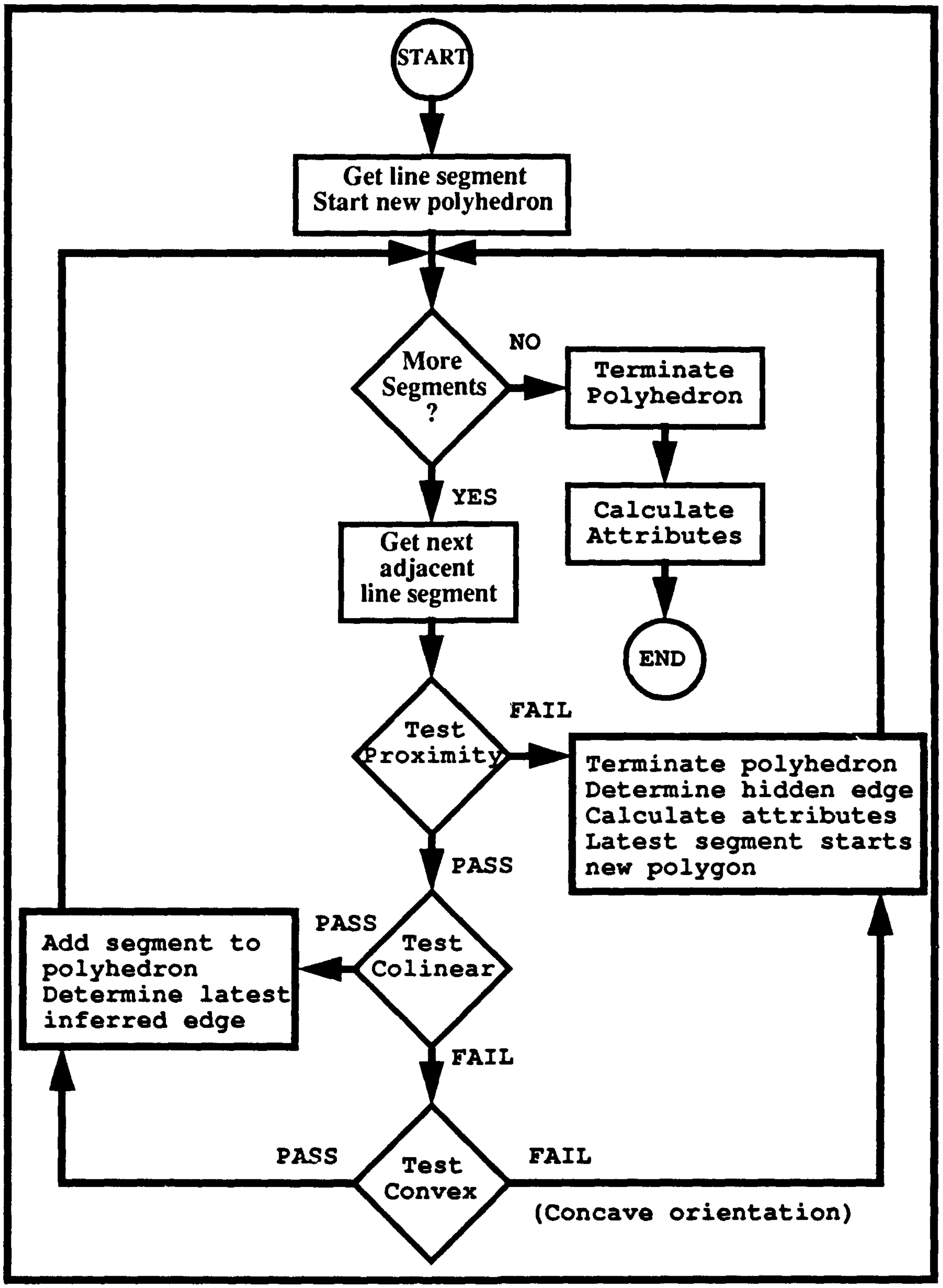


Figure 42: Algorithm to Build Polyhedra from Line Segments

edges. Figure 43 shows how the start point, regression line segments, inferred edges and hidden edge which make up a polyhedron cross-section define a series of triangular areas.

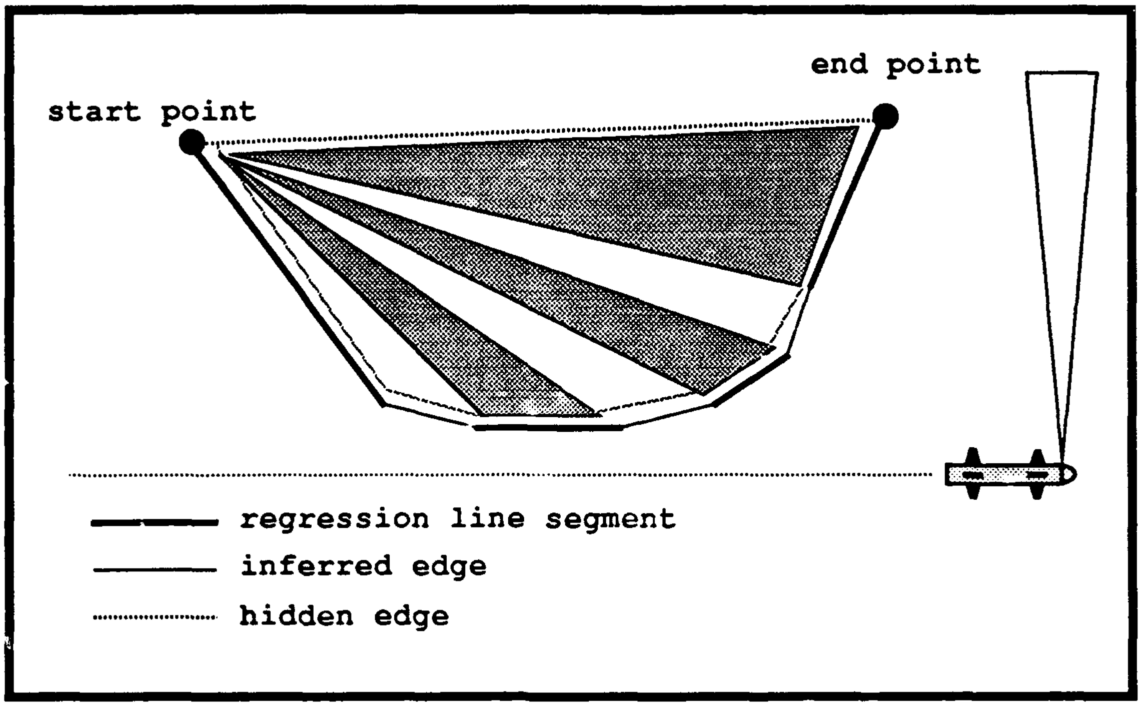


Figure 43: Summing Triangle Areas to Determine Polyhedron Cross-Sectional Area

Area of a single triangle is given by:

$$Aera\Delta = \frac{1}{2}[(X_2 - X_1)(Y_3 - Y_1) - (X_3 - X_1)(Y_2 - Y_1)].$$

Polyhedron cross-sectional area is determined by summing the area of these triangles, given by:

$$Area_{polyhedron} = \sum_{\substack{regression\\lines}} \left[Area\Delta_{\substack{start\_point,\\inferred\_edge}} + Area\Delta_{\substack{start\_point,\\regression\_line}} \right].$$

Centroid position for a triangle is calculated using:

$$\begin{matrix}Triangle\\ Centroid\end{matrix} = (X_C, Y_C) = (\frac{X_1+X_2+X_3}{3}, \frac{Y_1+Y_2+Y_3}{3}).$$

Centroid position for the polyhedron cross-section is precisely determined by taking the weighted average of each of the triangle centroids, given by:

$$\begin{matrix}Polyhedron\\ Cross-section\\ Centroid\end{matrix} = (\frac{Area_{\Delta_1}X_{C_1}+\ldots+Area_{\Delta_N}X_{C_N}}{Area_{polyhedron}}, \frac{Area_{\Delta_1}Y_{C_1}+\ldots+Area_{\Delta_N}Y_{C_N}}{Area_{polyhedron}}).$$

Polyhedron cross-section thinness is defined as the ratio of polyhedron area to the square of polyhedron perimeter, given by:

$$\begin{matrix}Polyhedron\\ Cross-section\\ Thinness\end{matrix} = \frac{PolyhedronArea}{(PolyhedronPerimeter)^2}.$$

If object height is needed and has not been directly measured, it can be estimated using heuristic rules based on object depth, bottom depth or independent object classification. Object volume is the product of cross-sectional area and measured or estimated object height.

Indirect attributes such as positional accuracy, confidence factor, inferred edges and hidden edges are also evaluated. Point positional accuracy is derived by combining current vehicle positional accuracy estimate with sonar accuracy or sonar beamwidth at the range to the object. Confidence factor can be defined independently of positional accuracy as a measure of how well the object matches a classification rule. Hidden edge length is a measure of what is not known about the object. Defining initial classification confidence factor as the ratio between hidden edge length and detected perimeter further indicates how much of the contact has actually been evaluated. Hidden edge metrics can be used to indicate whether further sonar investigation of the contact is desirable. Figure 44 shows detected edges, inferred edges and hidden edge relative to processed sonar returns, and how these geometric primitives may not fully reveal all features of a contact.

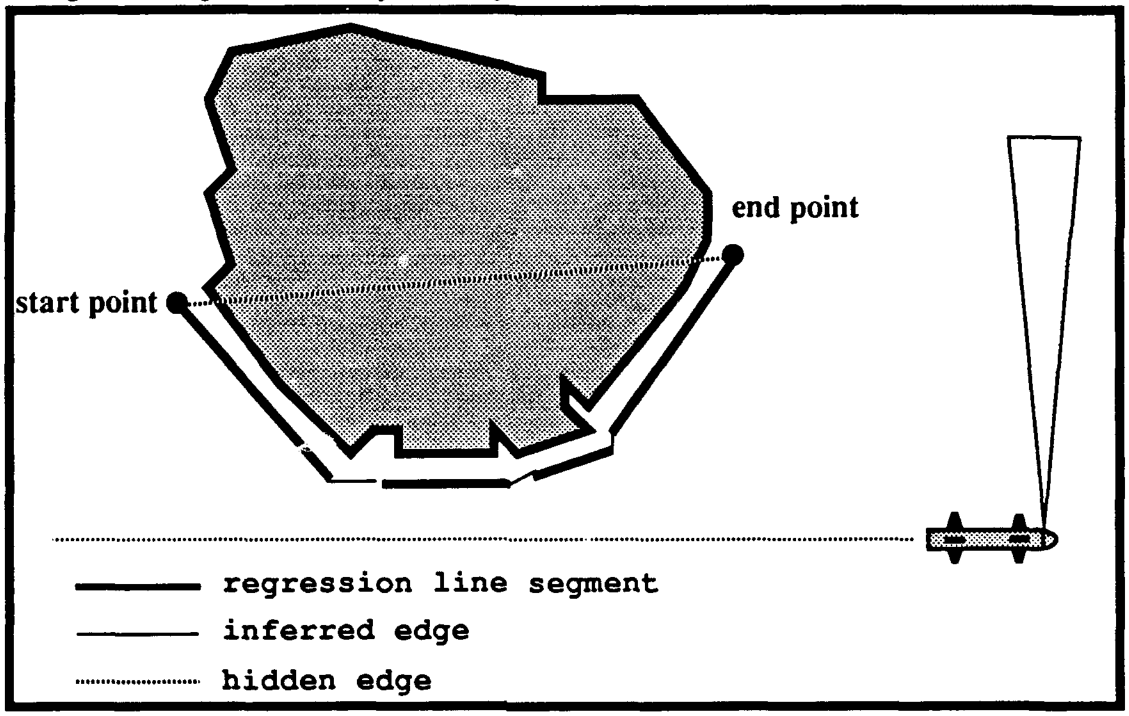


Figure 44: Polyhedron detected edges, inferred edges and hidden edge may not fully reveal all features of the sonar contact.

D. EXPERT SYSTEM HEURISTICS FOR SONAR CLASSIFICATION

While geometric analysis can be defined with mathematical precision, human knowledge regarding sonar classification is less rigorous and can best be encoded as expert system heuristics.

1. Classification Heuristics and Attribute Heuristics

Sonar classification is not always a well defined problem. For example, it is possible that sonar analysis of a single object can be performed from different directions and lead to completely different classifications. An analogy to classifying objects using simple range and bearing sonars is attempting to identify your surroundings while looking at the world through a steerable pinhole. It is difficult! Consequently, sonar classification criteria are often ambiguous and difficult to quantify, even when using formally derived geometric primitives. However, the heuristic approach used by expert systems is effective in many types of inexact problems and enables an autonomous system to obtain excellent sonar classification results.

Heuristics can be used for evaluating attributes such as object height when information is incomplete. Both attribute and classification heuristics can be easily modified in understandable ways despite the ambiguities of sonar analysis. The intuitive power of heuristics combined with the precision of geometric analysis gives sonar classification expert systems wide applicability and adaptability.

For this expert system, classification of sonar contacts is performed by comparing attributes of detected objects with predetermined attributes of known objects of interest. Different classification criteria are necessary and desirable for different environments. In particular, the different characteristics of deep ocean versus shallow water versus an artificial pool will constrain the possible types of objects to be detected. Knowledge of the current environment can be extremely useful when determining the specialized classification rules and heuristic criteria to be used for a given mission.

Precise classification of every possible object type may not be necessary for some missions. Resolution of an ambiguous classification typically requires multiple sensor looks, costing additional time and energy. Preliminary classification as a potential contact of interest may be sufficient to justify maneuvering for additional sensing and closer investigation. Conversely, objects deemed to be of no interest require no further investigation by the vehicle.

Size can be the primary classification attribute for most underwater objects of interest. However, size per se is not a strictly defined term. It is worth mention that significant object size may be indicated by a variety of attributes including cross-sectional area, volume, perimeter, thinness or hidden edge length. Any or all of these size-related attributes may require close evaluation in order to properly discriminate between similarly sized sonar targets such as mines and rocks.

2. Pattern-match Classification Examples

Examples of how heuristic rules work can illustrate how a sonar expert system can classify objects. Two examples are presented here.

Preliminary wall classification is possible during the execution of the polyhedron-building algorithm. Walls are defined as any flat linear surface of non-trivial length. Polyhedra being built can be considered as walls as long as each of the newly added regression line segments meet colinearity and proximity criteria. As soon as the polyhedron-building algorithm adds a new line segment based on convexity criteria, the polyhedron being built can be immediately reclassified from wall to object since the polyhedron is no longer linear.

Once a polyhedron has been built, all polyhedron attributes are automatically calculated. At this final stage, all of the preliminary work to quantitatively determine precise geometric objects greatly simplifies object classification. For instance, a polyhedron might be classified as a mine-like object whenever cross-sectional area is between 10 and 100 square feet (Figure 45). Other objects can be classified in an equally straightforward manner.

Some objects should not be uniquely classified. For example, discrimination between a scuba diver and a mine-like object may be difficult. A particular strength of the expert system approach is that each object can receive multiple classifications with associated confidence factors as appropriate. This feature allows high-level reasoning using uncertainty, rather than being constrained by an arbitrary and potentially incorrect single classification.

What was originally an intractable sonar classification problem is now much simpler and understandable at the highest level of the expert system.

3. Self-Diagnosis and Self-Correction

An additional strength of the expert system paradigm is that rules can be written to evaluate overall system performance, correcting internal vehicle problems without

```
(defrule classify-mine-like-object
; if this left-hand side of the rule is found to be true:
    ?poly <- (Polyhedron (status          COMPLETE)
                         (classification  OBJECT)
                         (start           ?startpolytime)
                         (end             ?endpolytime)
                         (area            ?area))

=>
; then perform this right-hand side of the classification rule:
    (if (and (>= ?area 10.0) (<= ?area 100.0)) ; area criteria test
     then (modify ?poly (classification MINE))
         (printout t "The polyhedron at times "    ?startpolytime
                                                   ?endpolytime)
         (printout t "has classification MINE.")
)
```

Figure 45: Classification Rule for a Mine-Like Object

external control. Self-diagnosis is possible when expert system evaluation of sensor data differs from *a priori* knowledge of the real world. Such differences can be automatically fed back into the system to correct the offending error. As an example, gyro error and gyro drift rate can be diagnosed and quantified when a deduced wall orientation does not match known geographic data. Updating system estimates of gyro error and gyro drift rate result in an immediate improvement in sonar accuracy and positional estimates.

E. EXPERT SYSTEM PARADIGM

The power of an expert system is essential for a sonar classifier to perform high-level reasoning using qualitative attribute and sonar classification heuristics. This section describes the salient features of expert systems which are pertinent to the development of an autonomous sonar classifier.

1. Expert System Characteristics

An expert system typically includes the following characteristics: it simulates human reasoning about a problem domain, it uses symbolic knowleuge representation and rules of thumb, it can analyze problems using heuristic or approximate methods which may not be guaranteed to succeed, and it deals with complexity which normally would require a human expert [Ref. 39]. Expert system development differs from usual software engineering approaches in that rules of thumb can be developed incrementally to solve large problems which do not necessarily have a clearly defined solution methodology. Complementary rules work together without explicit supervision to discover solutions, should any exist.

2. Knowledge Representation and Reasoning using Facts, Rules and an Inference Engine

Expert systems typically use facts to represent knowledge about the state of the problem domain. Facts can be known prior to execution as part of the problem definition, and can also be discovered during program execution as new data becomes available or new knowledge is deduced. Rules are heuristic representations of human reasoning that follow the condition-action model. If a rule finds certain conditions to be true, then corresponding actions will follow and the rule is said to execute or "fire". The inference engine is the mechanism that allows all of the rules to individually examine the fact database and fire. The order of rule precedence and firing may range from random sequencing to a strictly defined execution order.

Strict execution order is typical of traditional programming paradigms but is usually considered to be an undesirable constraint for expert systems. Interestingly, random firing of expert system rules often uncovers solutions to problems that might otherwise be considered unsolvable using a strictly defined sequential approach.

3. Rule Sets and Control of Execution Flow

Given that a single rule may be inadequate to fully evaluate a complex situation, often groups of rules called "rule sets" are written to work together on particularly difficult analysis tasks. Such organization of rules allows a manageable and modular approach to expert system design. However, the random nature of rule firing allowed by an inference engine may permit partially processed facts to be accessed and used by other rules before the original rule set has completed the group objective. For this reason it is usually desirable to ensure that rule sets are able to run to completion whenever activated, before other rules are again allowed to fire. As an example, implementation of the algorithm in Figure 42 requires several polyhedron building rule sets working together in a coordinated fashion with parametric regression rule sets.

Given the unpredictable nature of heuristics when solving highly complex problems, the expert system designer may need to impose some controls on execution flow among rule sets in order to ensure orderly execution. Randomness generally remains desirable and can still coexist within the bounds of rule execution flow control requirements.

4. Developing an Expert System

When a new application appears to be suitable for an expert system implementation, the first developmental step is to define the application problem in clearly understandable terms. This usually requires acquisition of expert knowledge in the problem area to be solved. The facts which may exist in the problem domain should be stated as unambiguously as possible. The overall problem should be logically grouped into simply stated subproblems consisting of condition-action rules.[Ref. 39]

Once the problem is well-defined, facts and rules are converted from plain language into the syntax of the expert system being used. When first building an expert system, facts and rules should be added in small numbers. Incrementally test the expert system and avoid adding new rule sets until examples show that existing rules work as intended. Additional adjustment may be necessary to ensure mutual rule cooperation whenever new rules are added. Such an incremental prototyping approach can be particularly effective when building large expert systems.[Ref. 40]

F. IMPLEMENTATION AND EVALUATION

1. NPS AUV Pool Experimentation

A distinct advantage of the NPS AUV program is that an actual vehicle specifically designed for maneuvering in a large swimming pool is available for test and evaluation. The sonar evaluation experiment presented in this chapter utilizes such actual vehicle sonar evaluation data. Video clips showing typical NPS AUV pool operations are available in [Ref. 41] and [Ref. 42].

2. CLIPS Expert System

A number of expert systems are commercially available. CLIPS ("C" Language Integrated Production System) was chosen for this application due to its portability, extensibility, capabilities, thorough documentation and interactive tutorials [Ref. 43]. CLIPS is also reasonably priced (approximately $450 or free for government agencies). CLIPS was developed by NASA to meet the varied requirements of NASA Mission Control Center delivery systems. CLIPS syntax is similar to the functional language Lisp and follows the if-then conditional rule model. The most recent versions of CLIPS add object-oriented and procedural programming capabilities.[Ref. 44] Since it is written in "C", CLIPS can run under most computer architectures. A feature of CLIPS that makes it particularly suitable for AUV use is that developed expert systems may be exported to nearly any microprocessor by autogeneration of executable "C" language code. CLIPS has an active user base, annual applications conferences, an applications abstract registry and is provided with complete source code.[Ref. 45]

3. NPS AUV Sonar Classification System

The program used to implement the concepts presented in this chapter was written using the CLIPS expert system. Actual sonar data collected by the NPS AUV is recorded in files for later use as input to the sonar classification expert system. This sonar data is analyzed off-line while running on a separate workstation.

A variety of outputs from the expert system provide several ways to visualize results. Two-dimensional graphics plots of raw sonar data and corresponding parametric regression line segments are shown on screen and as hard copy. An output file listing each individual geometric object and classification provides both hard copy of results and automatic input to the three-dimensional NPS AUV Integrated Simulator described below.

Sonar geometric analysis is computationally intensive. While running under the CLIPS environment on a Sun SPARC 2 workstation, the expert system is currently able to maintain a 7 Hz sonar return processing rate. This is nearly as fast as the 10 Hz data rate recorded by the NPS AUV and adequate for most real-time requirements. Optimization, elimination of network file server bottlenecks and source code compilation should further improve performance. Project goals include porting the NPS AUV sonar classification expert system to a microprocessor internal to the vehicle.

It is clear that a sonar classification expert system could operate autonomously in real time.

4. NPS AUV Integrated Simulator

Typically the development and testing of AUV hardware and software is greatly complicated by vehicle inaccessibility during operation. Integrated simulation remotely links vehicle components and support equipment with graphics simulation workstations. Integration of actual AUV components with three-dimensional simulation allows complete real-time, pre-mission, pseudo-mission and post-mission visualization and analysis in the lab. Integrated simulator testing of AUVs is a broad and versatile method that has proven very effective in the development of the NPS AUV sonar classification expert system [Ref. 14] [Ref. 42].

In particular, post-mission simulator playback of recorded telemetry, sonar sensor data and system state transitions supports in-depth reenactment, playback and analysis of processed sonar data. This scientific visualization approach permits rapid and precise development of geometric analysis techniques and classification heuristics for the NPS AUV sonar classification system.

High-resolution three-dimensional graphics workstations can provide real-time representations of vehicle dynamics, control system behavior, mission execution, sonar processing and object classification. Use of well-defined, user-readable mission log files as the data transfer mechanism allows consistent and repeatable simulation of all AUV operations.

G. EXPERIMENTAL RESULTS

1. Classification Test Scenario

An example best demonstrates successful classification of actual sonar returns. A single swimmer was chosen to represent a mine-like object and was positioned as a target near the right-hand wall of the NPS swimming pool, shown in Figure 46.

The NPS AUV was programmed to follow a racetrack traversal of the pool and record all pertinent data. Figure 47 shows individual left transducer sonar returns plotted as circles and vehicle track as a large oval, while the line segments calculated by the parametric regression algorithm are shown superimposed. Some distortion is evident due to unmodeled sideslip error in vehicle track data.

2. Experimental Results

The sonar data recorded by the NPS AUV in the pool are uploaded after mission completion via modem and processed off-line by the sonar classification expert system. Classification results are then graphically rendered by the NPS AUV Integrated Simulator running on a *Silicon Graphics Iris workstation*. This three-dimensional display shows all

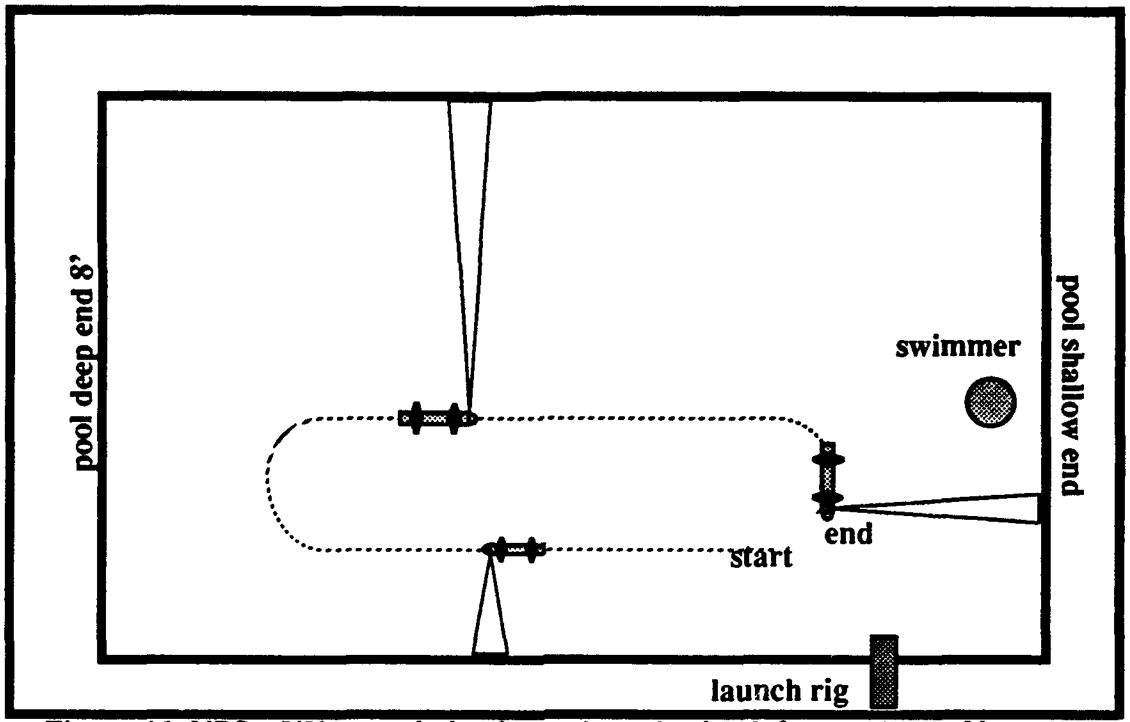


Figure 46: NPS AUV test mission is conducted using left transducer. Note the swimmer target.

generated parametric regression line segments, inferred edges, hidden edges, and detected walls. The overall pool graphics display as seen from a viewpoint high above the pool is shown in Figure 48. The target of interest met classification criteria for a mine-like object and a simulation close-up is shown in Figure 49.

The integrated simulator has the additional feature of being able to play back sonar detections and classifications simultaneously with vehicle motion in real time or slow motion. Evaluation of sonar classification results using the scientific visualization techniques provided by the integrated simulator was extremely helpful during development and testing of sonar expert system classification heuristics.

The assessment of experimental results is that the NPS AUV Autonomous Sonar Classification System is highly effective at classifying objects despite the low resolution of the active sonar employed.

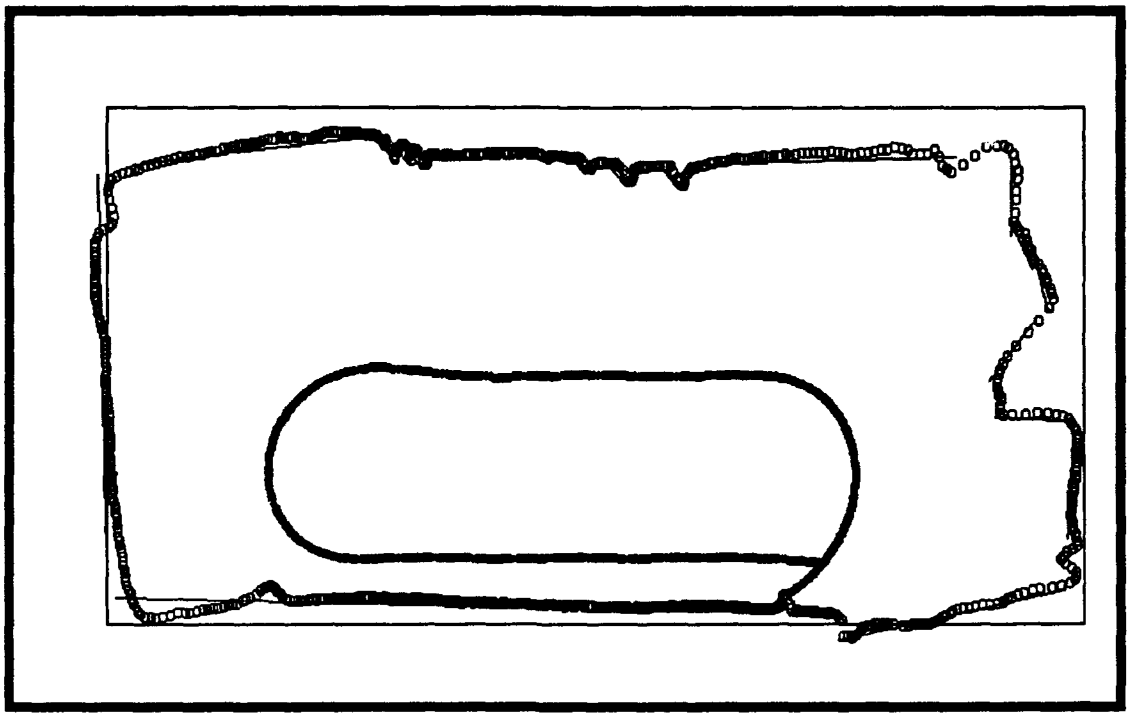


Figure 47: NPS AUV Sonar Classification Expert System Plot of Pool Data and Parametric Regression Line Segments

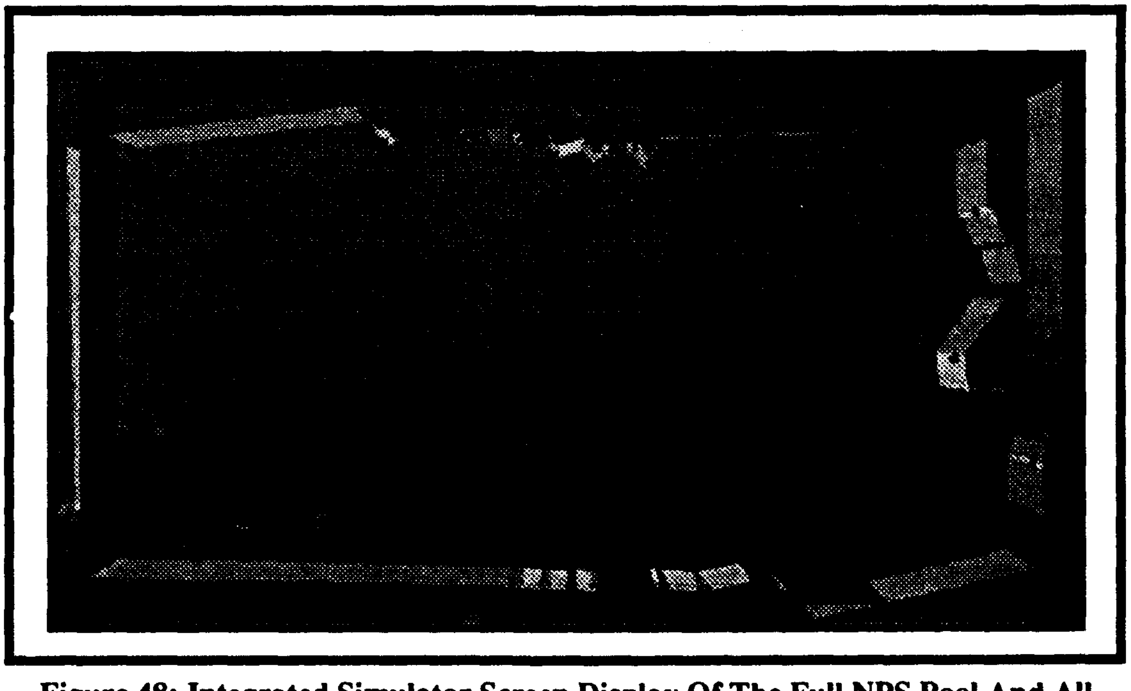


Figure 48: Integrated Simulator Screen Display Of The Full NPS Pool And All Sonar Classifications

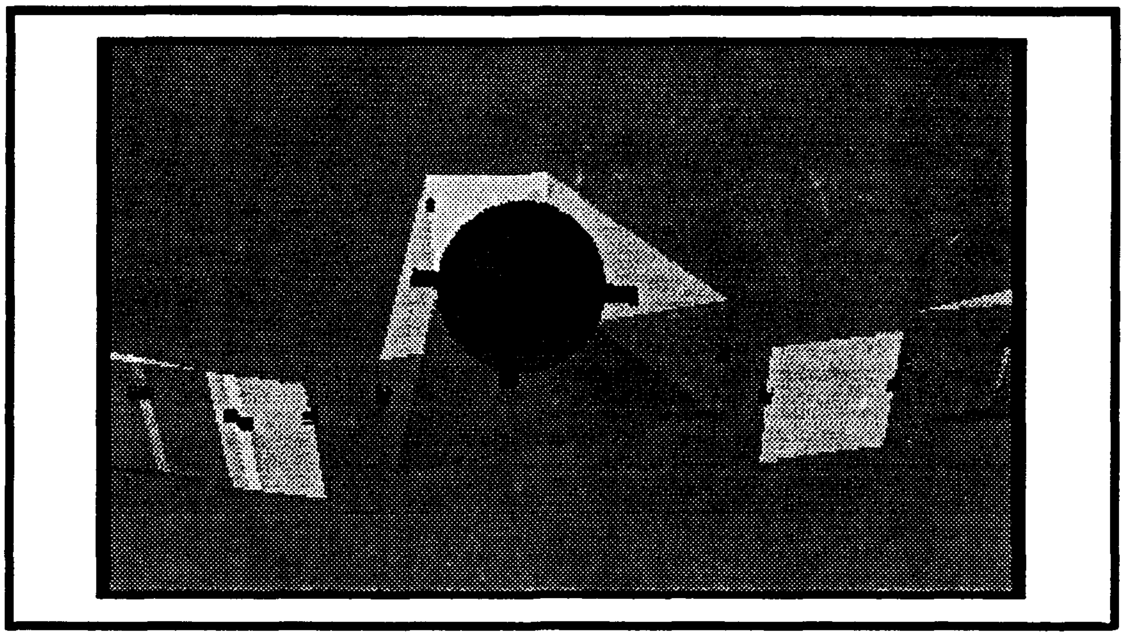


Figure 49: Integrated Simulator Display Close-Up of a Mine-Like Object Classified by the Sonar Expert System using Directed Edges, Inferred Edges, Hidden Edges and Cross-Sectional Area

H. DISCUSSIONS AND APPLICATIONS

1. Extendabi'''ty to Video, Lasers, Complex Sonars and Sensor Fusion

Active sonar is not the exclusive sensor used for underwater object detection and classification. A variety of other sensors are coming into use including underwater videocameras and lasers. In addition to range and bearing data, advanced sonars may provide completely different types of data such as frequency spectra, doppler or long-range conical beam data. Ultrasonic sonars have also been employed by land vehicles.

All of these sensors share common characteristics which allow autonomous analysis by expert systems. Each sensor type provides data sets that can be analyzed using geometric reasoning techniques. In every case expert knowledge can define both quantitative and heuristic rules for processing sensor outputs to create primitive geometric objects, thus allowing object classification.

Sensor fusion is the correlation of multisource information to resolve ambiguity and increase confidence in individual classifications. Sensor fusion is particularly valuable in offsetting the weaknesses of one sensor type with the strengths of another. An example of sensor fusion might be to correlate accurate laser bearing data with accurate sonar range data. A thorough survey on multisensor fusion roles, approaches and applications is provided by [Ref. 32]. Sensor fusion can be directly implemented using the pattern-matching capabilities of a multisensor classification expert system.

2. Intelligent Remote Sensors

The use of remote sensors is becoming commonplace. The primary limitation of most remote sensors is that they have little ability to independently react to sensor inputs. Most sensing devices require direct control or have an arbitrary sampling period, while continuously-sensing devices require dedicated data communication lines. Remote underwater sensors need to operate autonomously or with minimal external control in order to improve their efficiency, capabilities and cost-effectiveness. Embedding an expert system application using microprocessor-based control is a feasible method to create intelligent and autonomous remote sensors.

3. Data Reduction

Most sensor data is high bandwidth. Autonomous vehicles, remotely operated vehicles and remote sensors typically receive extremely large amounts of data. Storage or transmission of raw data for off-line processing is undesirable and imposes unreasonable memory capacity and communications requirements. A significant benefit of autonomous classification is that it reduces massive amounts of raw data into concise information which can be efficiently recorded or communicated. Data without value is easily filtered. The overall data compression ratio can equal several orders of magnitude.

4. Future Use of Expert Systems by Autonomous Vehicles

If autonomous vehicle sensors and missions are to become increasingly capable and sophisticated, it is likely that parallel processing of distributed artificial intelligence modules will be necessary in order to provide adequate computing power with real-time response. A typical set of high-level processes might include detection and classification for multiple sensors, path planning, search, systems control and others. None of these processes is completely independent, but typically each process can run in parallel with the others most of the time. One abstract software architecture that supports such a distributed approach is the blackboard paradigm.

A blackboard system directly extends the functionality seen in an expert system for a collection of distributed processes [Ref. 39]. A good metaphor for the blackboard approach is a group of human experts working together on a large problem using a blackboard as their means of communication. Problem definition, data, questions and answers can all be written and read on various sections of the blackboard. Each independent expert has full access to the blackboard and looks for information pertinent to his area of expertise. When an expert develops some result or new question worth communicating to the group, that information is recorded on the blackboard.

Similarly, a blackboard system has distributed independent knowledge sources, each of which can use any method desired to solve portions of a large problem. Communications are recorded on the blackboard and are available to all knowledge sources. Complex problems are solved through cooperative reasoning.[Ref. 46] As another example, each of the processes shown in Figure 37 could be implemented as separate knowledge sources for a blackboard. Expert systems are well suited as knowledge sources for the blackboard paradigm.

Development of autonomous expert systems is likely to provide intelligent components that will remain useful in the advanced architectures of future autonomous vehicles.

I. CONCLUSIONS AND FUTURE DEVELOPMENT

Autonomous sonar classification systems can accurately detect and classify objects in the underwater environment. Precise geometric analysis is combined with qualitative expert system heuristics to provide a flexible and robust approach with wide applicability. Autonomous classification systems are capable of supporting sophisticated real-time applications in working autonomous vehicles.

Future developments should include additional higher-level heuristic rules for refining object classifications. Because AUV pool missions are limited in scope it is also important that AUV sonar simulation beyond the work completed by C. Floyd be conducted. Analysis of rudimentary virtual sonar data should be possible within the graphic simulator environment [Ref. 6]. Future development should also investigate the possible synergistic effects of combining the features of expert systems with those of neural networks. A neural network which is trained by an expert system in real-world environments could be employed on the microchip level in future AUVs.

In conclusion, the application of autonomous sonar contact classification systems is critical for the successful conduct of important Naval missions such as the minefield search mission presented in this thesis.

VII. CONCLUSION

This thesis addressed the following important aspects of the AUV minefield search mission:

- minefield search path planning and replanning
- autonomous sonar classification of underwater objects using expert systems
- AUV software performance evaluation through graphic simulation

The conclusions from each of these separate mission aspects are addressed at the end of their respective chapters.

The involvement of officers at the Naval Postgraduate School in AUV research provides an important link between the laboratory and the "real world" where AUVs will be employed. The cooperation between officers and researchers who mutually develop this technology will ensure that AUVs meet fleet needs in the most effective manner. It is hoped that this thesis has contributed to this goal.

APPENDIX A: TWO-DIMENSIONAL MINEFIELD SEARCH SOURCE CODE

```
/*************************************************************************
Title:  mine2d.h
Author: Mark Compton
Course: Thesis
Date:   09 March 1992

Description: This program is a graph-based, two-dimensional search algorithm
             for minefield search missions by AUVs.
Support:     mine2d.c
Display:     Output files may be run in Two-Dimensional Graph Search Tool.
*************************************************************************/

/*Preprocessing Directives**********************************************/
#define BOXWIDTH  15.0  /* box width                 */
#define BOXHEIGHT 15.0  /* box height                */
#define MAXX 63         /* number of nodes in x direction */
#define MAXY 63         /* number of nodes in y direction */
#define SUBX 7          /* number of subareas in x direction */
#define SUBY 7          /* number of subareas in y direction */
#define SUBXNODES 9     /* number of nodes in each subarea in x direction */
#define SUBYNODES 9     /* number of nodes in each subarea in y direction */
#define MAXSTRING 20    /* for file names */

#define ODD  1          /* defines if a value is odd */
#define EVEN 0          /* defines if a value is even */

#define TRUE  1         /* needed on non iris */
#define FALSE 0

/* defines states of a node */
#define FREE      0     /* nothing at node */
#define OBSTACLE 1      /* obstacle at node */
#define AUV       2     /* vehicle (Autonomous Underwater Vehicle) */
#define ADJACENT 3      /* node adjacent to vehicle */
#define ACTIVE    4     /* detected, non-object, not visited */
#define VISITED  5      /* node has been previously visited */
#define ASPATH    6     /* local path selected by A-Star search */

/* defines states of previous node array nodes */
#define NOTEVAL  0      /* node not evaluated - default state */
#define FRONTIER 1      /* node placed on agenda (visit list) */
#define CHECKED  2      /* node evaluated and taken off agenda */

/* defines direction of vehicle */
#define NODIR 0
#define N     1
#define NE    2
#define E     3
#define SE    4
#define S     5
#define SW    6
#define W     7
#define NW    8

/*Globals***************************************************************/
FILE *obstacleifp;           /* pointer for receiving obstacles from file */
FILE *missionofp;           /* pointer for sending mission to file */
char inobstacle[MAXSTRING];  /* input file name for obstacles */
char outmission[MAXSTRING];  /* output file name for mission */
int searchmethod = 0;        /* select search method */
int searchtype = 0;            /* select search type */
int auvstartx = 0;             /* starting position for AUV */
int auvstarty = 0;             /* starting position for AUV */
int verbose = FALSE;           /* flag to control verbose output */
int storemissiondata = FALSE;  /* flag for sending mission data to file */
int apriorobstacle = FALSE;    /* shows obstacles prior to mission viewing */
int retrievedata = FALSE;      /* flag for retrieving obstacle data */
int auvsteps = 0;              /* number of graph steps taken by vehicle */
float distrav = 0.0;           /* total distance traveled by vehicle */
int obstaclecnt = 0;           /* first time visited count */

/*Structures***********************************************************/
struct location          /* basic structure for location of an entity */
{
  int xpos,ypos;         /* grid coordinates */
};

typedef struct location Location;

struct node              /* basic structure of a search area node */
{
  Location grid;         /* grid coordinates of node */
  int subarea;           /* subarea node is located in */
  double x,y;            /* world position of node */
  int dir;               /* direction node is looking */
  int state;             /* status of node */
  int visited;           /* node previously visited by vehicle flag */
  struct node *prior,    /* pointer to prior node in subarea */
              *next;     /* pointer to next node in subarea */
};
  typedef struct node Node;

Node Wld[MAXX][MAXY];    /* 2-D array for storing search world nodes */

struct vehicle           /* AUV */
{
  Location grid;         /* grid coordinates of Vehicle */
  int subarea;           /* subarea vehicle is located in */
  double x,y;            /* world position of vehicle */
  int dir;               /* direction vehicle is looking or traveling */
};
  typedef struct vehicle Vehicle;

Vehicle Auv;             /* Autonomous Underwater Vehicle */

struct activenode;       /* announce future structure */

struct activesubarea     /* structure of subarea in visit list */
{
  int subareanum;                        /* number of the subarea */
  struct activenode *nodelistptr;        /* pointer to first node in list */
  struct activesubarea *nextsubareaptr;  /* pointer to next subarea in list */
};
  typedef struct activesubarea Activesubarea;
  typedef Activesubarea *SUBAREALINK;    /* pointer to Activesubarea */
  SUBAREALINK zh;                        /* pointer to first subarea in list */

struct activenode        /* structure of node in visit list */
{
  int x,y;                            /* coordinates of node */
  struct activenode *nextnodeptr;     /* pointer to next node in list */
};
  typedef struct activenode Activenode;
  typedef Activenode *NODELINK;       /* pointer to Activenode */

struct asnode            /* structure of node in a_star search agenda list */
{
  int x,y;                            /* coordinates of asnode */
  float cumcost;                   /* cum cost from start to asnode */
  float criteria;                  /* sum of cumcost and eval function */
  struct asnode *nextasnodeptr;    /* points to next asnode in list */
};
  typedef struct asnode Asnode;
  typedef Asnode *ASNODELINK;      /* pointer to Asnode */
  ASNODELINK ash;                  /* points to head of list */

struct asvertex             /* structure for node in previous node array */
{
  int xprev,yprev;          /* coordinates of previous vertex */
  float cumvertcost;        /* cumulative cost from start */
  int condition;            /* state of node */
  ASNODELINK ptrtoagenda;   /* points to corresponding node in agenda */
};
  typedef struct asvertex Asvertex;
  Asvertex Map[MAXX][MAXY];

struct newpath                     /* structure for path to goal */
{
  int xpath,ypath;                 /* coordinates of nodes in path */
  struct newpath *nextnewnode;     /* pointer to next newpath node */
};
  typedef struct newpath Newpath;
  typedef Newpath *NEWPATHLINK;    /* pointer to Newpath */

/*functions***************************************************************/
/* minefield search functions */
void set_up();
void build_world();
int subarea();
void retrieveobstacledata();
void start_position();
void conduct_mission();
void ladder_search();
void perform_search_routine();
void analyze_adjacent_nodes();
int node_visible();
void add_to_visit_list();
SUBAREALINK insert_subarea();
void add_to_node_list();
void delete_from_visit_list();
SUBAREALINK locate_subarea();
int delete_from_node_list();
void delete_from_subarea_list();
NODELINK get_priority_node_from_visit_list();
NODELINK determine_best_direction();
void advance_vehicle_and_delete_path();

/* A-Star functions */
NEWPATHLINK search_a_star();
void analyze_neighbor_nodes_a_star();
void add_to_agenda_list();
void delete_agenda_node();
ASNODELINK get_principle_node_and_delete_from_agenda();
void delete_agenda_list();
void mark_vertex_path();
NEWPATHLINK create_path_list();

/* General functions */
int odd();
float compute_dist();
int inside_area();
int is_adjacent();
int adjacent_node();
void clean_up();
void test_advance();
void print_active_list();
void print_agenda_list();
void print_path_list();

/****************************************************************************
Title:  mine2d.c
Author: Mark Compton
Course: Thesis
Date:   09 Mar 92

Description: This program conducts selected searches of a simulated
             minefield for Autonomous Underwater Vehicles.
****************************************************************************/

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "mine2d.h"

main ()
{

  int testrun = TRUE;

  set_up();
  conduct_mission();
  clean_up();

}

/***************************************************************************/
/*SET_UP..Get user inputs                                                  */
/***************************************************************************/
void set_up()

{

  char answer = 'n';
  char reply  = 'n';
  char ack    = 'n';

  /* Ask if verbose output to screen */
  printf("\nDo you wish verbose output to screen?  ");
  scanf("%c",&reply);
  if (reply == 'y' || reply == 'Y')  verbose = TRUE;
  reply = 'n';

  /* Setup for retrieving obstacle data */
  scanf("%c",&answer); /* hack to clear carrage return from buffer */
  printf("\n\nObstacle data may be retrieved from a file. \n\n");
  printf("Will you be retrieving data from a file?  ");
  scanf("%c",&answer);
  if (answer == 'y' || answer == 'Y')
  {
    retrievedata = TRUE;
    printf("\n\nPlease enter the obstacle input file name: \n");
    scanf("%s",inobstacle);
  }

  /* Setup for sending mission data to file */
  scanf("%c",&reply); /* hack to clear carrage return from buffer */
  printf("\nMission data may be saved to a file. \n\n");
  printf("Will you be sending mission data to a file?  ");
  scanf("%c",&reply);
  if (reply == 'y' || reply == 'Y')
  {
    storemissiondata = TRUE;
    printf("\n\nPlease enter the mission file name: \n");
    scanf("%s",outmission);
    missionofp = fopen(outmission,"w");   /* open the mission file */
  }

  /* Setup for showing obstacles prior to viewing mission playback */
  scanf("%c",&ack); /* hack to clear carrage return from buffer */
  printf("\nObstacles may be displayed prior to viewing mission. \n\n");
  printf("Do you wish to view obstacles prior to viewing mission?  ");
  scanf("%c",&ack);
  if (ack == 'y' || ack == 'Y')
  {
    apriorobstacle = TRUE;
  }

  /* Select search method */
  printf("\nSelect search method from the following file by typing \n");
  printf("the desired number:  ");
  printf("\n\n1. Ladder with Sub-area priorities.");
  printf("\n2. Ladder without Sub-area priorities. ");
  printf("\n3. None.\n");
  scanf("%d",&searchmethod);

  /* Select search type */
  if(searchmethod == 1 || searchmethod == 2)
  {
    printf("\nSelect search type from the following file by typing \n");
    printf("the desired number:  ");
    printf("\n\n1. A Star.");
    printf("\n2. None. ");
    printf("\n3. None. ");
    printf("\n4. Test Advance. \n");
    scanf("%d",&searchtype);
  }

  /* Select AUV starting position */
  printf("\nSelect the start position for the AUV (integer value): \n");
  printf("\nx =  ");
  scanf("%d",&auvstartx);
  printf("\ny =  ");
  scanf("%d",&auvstarty);

  build_world();  /* initialize array and node structure */

}

/***************************************************************************/
/*BUILD_WORLD..Builds the world for vehicle operations                     */
/***************************************************************************/
void build_world()

{

  int i,j;  /* variables for rows and columns */

  for(j=0; j < MAXY; j=j+1)
  {
    for(i=0; i < MAXX; i=i+1)
    {
      Wld[i][j].x = i;
      Wld[i][j].y = j;
      Wld[i][j].state = FREE;
      Wld[i][j].visited = FALSE;
      Wld[i][j].grid.xpos = i;
      Wld[i][j].grid.ypos = j;
      if (searchmethod == 2)
      {
        Wld[i][j].subarea = 1;
      }
      else
```

```
        {
          Wld[i][j].subarea = subarea(i,j);
        }
/* following commented out due to excessive verbose time consumed */
/*      if(verbose == TRUE)
          printf("\nWorld Node is %d,%d",
                     Wld[i][j].grid.xpos,Wld[i][j].grid.ypos);
*/
      }
    }
  if (retrievedata == TRUE)
  {
    retrieveobstacledata();    /* fill in obstacles */
  }
    start_position();          /* initiates AUV start position */

}

/***************************************************************************/
/*SUBAREA..Determines which subarea a node is in                          */
/***************************************************************************/
int subarea(a,b)

int a,b; /* feed in the grid coordinates */

{

  static int i,j,row,column,subarea;

  for (i = 1; i <= SUBY; ++i)                /* step thru subarea rows */
  {
    for (j = 1; j <= SUBX; ++j)              /* step thru subarea columns */
    {
      if (b >= (i-1)*SUBYNODES && b < i*SUBYNODES)
      {
        row = i;
      }
      if (a >= (j-1)*SUBXNODES && a < j*SUBXNODES)
      {
        column = j;
      }
    }
  }

  if (odd(row))
  {
    subarea = ((row-1)*SUBY)+column;
  }
  else
  {
    subarea = ((row*SUBY)+1)-column;
  }
  /* verbose output commented out, too verbose */
  /*if(verbose == TRUE)
      printf("\nSubarea is %d",subarea);
  */
  return subarea;
}

/***************************************************************************/
/*RETRIEVEOBSTACLEDATA..Retrieves obstacle data from file if requested    */
/***************************************************************************/
void retrieveobstacledata()

{

  int x,y;
```

```
    obstacleifp = fopen(inobstacle,"r");
    /* read obstacle positions from obstacle file */
    while(fscanf(obstacleifp, "%d%d", &x,&y) != EOF)
    {
      obstaclecnt = obstaclecnt + 1;
      Wld[x][y].state = OBSTACLE;
      if(verbose == TRUE)
        printf("\nObstacle at %d,%d",x,y);
      /* send obstacle world data to mission file */
      /* note that % indicates world data */
      if(storemissiondata == TRUE && aprioriobstacle == TRUE)
      fprintf(missionofp,"%c %d %d %d %d %f %d %f %f\n",'%',
                                              Wld[x][y].grid.xpos,
                                              Wld[x][y].grid.ypos,
                                              Wld[x][y].dir,
                                              Wld[x][y].state,
                                              distrav,auvsteps,0.0,0.0);
    }
    fclose(obstacleifp);

  }

  /***************************************************************************/
  /*START_POSITION..Defines the starting position of the AUV                 */
  /***************************************************************************/
  void start_position()

  {

    /* read start position from global variable */
    Auv.x = Wld[auvstartx][auvstarty].x;
    Auv.y = Wld[auvstartx][auvstarty].y;
    Auv.grid.xpos = Wld[auvstartx][auvstarty].grid.xpos;
    Auv.grid.ypos = Wld[auvstartx][auvstarty].grid.ypos;
    Auv.dir = NODIR;
    Wld[auvstartx][auvstarty].state = AUV;
    Wld[auvstartx][auvstarty].visited = TRUE;
    if(verbose == TRUE)
      printf("\nAUV start = %d,%d",Auv.grid.xpos,Auv.grid.ypos);
    /* send AUV start position to file */
    /* note that $ indicates search path data */
    if (storemissiondata == TRUE)
    {
    fprintf(missionofp,"%c %d %d %d %d %f %d %f %f\n",'$',
                                  Wld[auvstartx][auvstarty].grid.xpos,
                                  Wld[auvstartx][auvstarty].grid.ypos,
                                  Wld[auvstartx][auvstarty].dir,
                                  Wld[auvstartx][auvstarty].state,
                                  distrav,auvsteps,0.0,0.0);

    /* clean up old AUV position and send to file */
    /* $ represents vehicle data */
    fprintf(missionofp,"%c %d %d %d %d %f %d %f %f\n",'$',
                                  Wld[auvstartx][auvstarty].grid.xpos,
                                  Wld[auvstartx][auvstarty].grid.ypos,
                                  NODIR,
                                  VISITED,
                                  distrav,auvsteps,0.0,0.0);
    }

  }

  /***************************************************************************/
  /*CONDUCT_MISSION..Initiates selected mission                              */
  /***************************************************************************/
  void conduct_mission()

  {
```

```
  if(searchmethod == 1)
  {
    ladder_search();
  }
  if(searchmethod == 2)
  {
    ladder_search();
  }
  if(searchmethod == 3)
  {
    if(verbose == TRUE)
    printf("\nThis search method is not yet programmed.\n");
  }

}

/****************************************************************************/
/*LADDER_SEARCH..Conducts modified ladder search                            */
/* Logic as follows:
    1. Look at nodes adjacent to vehicle to determine their status.
       a. If nodes have not been visited and are not obstacles and
             are not currently on the visit list, add them to the
             visit list.
    2. Determine the highest priority node.
       a. Highest priority node is in list of highest priority subarea.
       b. Within highest priority subarea, determine closest node to vehicle.
       c. If more than one closest node, choose horizontal over vertical,
          right over left, up over down.
    3. Move vehicle to priority node.
    4. Delete visited node from visit list.
    5. Search complete when visit list is empty.
*/

/****************************************************************************/
void ladder_search()

{

  float mindistpos,eratio,estepratio;

  NODELINK visitnode = NULL; /* priority node to visit from visit list */

  analyze_adjacent_nodes();
  visitnode = get_priority_node_from_visit_list();
  if(verbose == TRUE)
    printf("\nThe priority node is %d,%d\n",visitnode->x,visitnode->y);
  while (visitnode != NULL)
  {
    perform_search_routine(visitnode); /* moves AUV to priority node */
    analyze_adjacent_nodes();
    visitnode = get_priority_node_from_visit_list();
    if(verbose == TRUE)
      printf("\nThe priority node is %d,%d",visitnode->x,visitnode->y);
  }
  if (visitnode == NULL)
  {
    /* compute final statistics */
    mindistpos = ((MAXX * MAXY) + MAXY) - (2 * obstaclecnt);
    eratio = mindistpos / distrav;
    estepratio = (((MAXX * MAXY) + MAXY) - obstaclecnt)/(auvsteps);
    fprintf(missionofp,"%c %d %d %d %d %f %d %f %f\n",'$',
                       Auv.grid.xpos,Auv.grid.ypos,0,AUV,
                       distrav,auvsteps,
                       eratio,estepratio);

    printf("\nSEARCH COMPLETE!!\n");
  }
```

```
}

/****************************************************************************/
/*PERFORM_SEARCH_ROUTINE..This selects the type of search routine used to*/
/*go to the selected goal node, conducts search and returns path         */
/****************************************************************************/
void perform_search_routine(goalnodeptr)

  NODELINK goalnodeptr; /* node to be visited from AUV current posit */

{

  NEWPATHLINK pathlistptr; /* points to list giving vehicle path */

  if (searchtype == 1)
  {
  if(verbose == TRUE)
    printf("\n\nBeginning A-star search to next goal node\n\n");
    /* determine path to goal from AUV current position with A-star */
    pathlistptr = search_a_star(goalnodeptr);
    /* move vehicle along path, replan if obstacle encountered */
    /* and delete path list when finished */
    advance_vehicle_and_delete_path(pathlistptr);
    free(pathlistptr);
    if(verbose == TRUE)
      printf("\n\nEnding A-star search ... reached this goal\n\n");
  }
  if (searchtype == 2) printf("\nNot a usable search!!\n");
  if (searchtype == 3) printf("\nNot a usable search!!\n");
  if (searchtype == 4)
  {
    if(verbose == TRUE)
      printf("\nThe search routine is test advance\n");
    test_advance(goalnodeptr);
  }

}

/****************************************************************************/
/*ANALYZE_ADJACENT_NODES..Determine status of nodes adjacent to AUV      */
/*if nodes is not an obstacle and has not been visited then add it to    */
/*the visit list                                                         */
/****************************************************************************/
void analyze_adjacent_nodes()
{

  /* NORTH */
  /* check adjacent node */
  if(Wld[Auv.grid.xpos][Auv.grid.ypos+1].state == OBSTACLE &&
     storemissiondata == TRUE)
  {
    fprintf(missionofp,"%c %d %d %d %d %f %d %f %f\n",'%',
      Wld[Auv.grid.xpos][Auv.grid.ypos+1].grid.xpos,
      Wld[Auv.grid.xpos][Auv.grid.ypos+1].grid.ypos,
      Wld[Auv.grid.xpos][Auv.grid.ypos+1].dir,
      Wld[Auv.grid.xpos][Auv.grid.ypos+1].state,
      distrav,auvsteps,0.0,0.0);
  }
  if((node_visible(Auv.grid.xpos,Auv.grid.ypos+1) == TRUE) &&
    (Wld[Auv.grid.xpos][Auv.grid.ypos+1].visited == FALSE) &&
    (Wld[Auv.grid.xpos][Auv.grid.ypos+1].state != ACTIVE)  &&
    (Wld[Auv.grid.xpos][Auv.grid.ypos+1].state != OBSTACLE))
  {
    if(verbose == TRUE)
      printf("\nAnalyzing adjacent node %d,%d",
                                          Auv.grid.xpos,Auv.grid.ypos+1);
    add_to_visit_list(Auv.grid.xpos,Auv.grid.ypos+1);
  }

                                        .
                                        .
                                        .
                                        .

  /* NORTHWEST */
  /* check adjacent node */
  if(Wld[Auv.grid.xpos-1][Auv.grid.ypos+1].state == OBSTACLE &&
     storemissiondata == TRUE)
  {
    fprintf(missionofp,"%c %d %d %d %d %f %d %f %f\n",'%',
      Wld[Auv.grid.xpos-1][Auv.grid.ypos+1].grid.xpos,
      Wld[Auv.grid.xpos-1][Auv.grid.ypos+1].grid.ypos,
      Wld[Auv.grid.xpos-1][Auv.grid.ypos+1].dir,
      Wld[Auv.grid.xpos-1][Auv.grid.ypos+1].state,
      distrav,auvsteps,0.0,0.0);
  }
  if((node_visible(Auv.grid.xpos-1,Auv.grid.ypos+1) == TRUE) &&
    (Wld[Auv.grid.xpos-1][Auv.grid.ypos+1].visited == FALSE) &&
    (Wld[Auv.grid.xpos-1][Auv.grid.ypos+1].state != ACTIVE)    &&
    (Wld[Auv.grid.xpos-1][Auv.grid.ypos+1].state != OBSTACLE))
  {
    if(verbose == TRUE)
      printf("\nAnalyzing adjacent node %d,%d",
                                     Auv.grid.xpos-1,Auv.grid.ypos+1);
    add_to_visit_list(Auv.grid.xpos-1,Auv.grid.ypos+1);
  }

}

/*************************************************************************/
/*NODE_VISIBLE..Determines if a node adjacent to AUV is inside of the    */
/*search area and is available                                           */
/*************************************************************************/
int node_visible(x,y)

int x,y;

{

  if((inside_area(x,y)) && (Wld[x][y].state == FREE))
    return TRUE;
  else
    return FALSE;

}

/*************************************************************************/
/*ADD_TO_VISIT_LIST..Adds node to visit list                             */
/*************************************************************************/
void add_to_visit_list(xpos,ypos)

int xpos,ypos;

{

  SUBAREALINK subareaheadptr = NULL;    /* temp ptr to first subarea */
  SUBAREALINK currsubareaptr = NULL;    /* temp ptr to current subarea */
  SUBAREALINK subareaofinterest = NULL; /* ptr to subarea of interest */
  int currentsubarea;                /* indicates subarea of interest number */

  currentsubarea = Wld[xpos][ypos].subarea; /* subarea of interest */
  Wld[xpos][ypos].state = ACTIVE;
  /* send active node to output file */
  if (storemissiondata == TRUE)
  {
    fprintf(missionofp,"%c %d %d %d %d %f %d %f %f\n",'$',
                                Wld[xpos][ypos].grid.xpos,
                                Wld[xpos][ypos].grid.ypos,
                                NODIR,
                                Wld[xpos][ypos].state,
                                distrav,auvsteps,0.0,0.0);
    }

    if (zh == NULL)  /* subarea list is empty so create subarea and first node */
    {
      subareaheadptr = (Activesubarea*)malloc(sizeof(Activesubarea));
      zh = subareaheadptr;        /* attach start pointer to first subarea record */
      subareaheadptr->subareanum = currentsubarea;
      subareaheadptr->nodelistptr = (Activenode*)malloc(sizeof(Activenode));
      subareaheadptr->nodelistptr->x = xpos;
      subareaheadptr->nodelistptr->y = ypos;
      subareaheadptr->nodelistptr->nextnodeptr = NULL;
      subareaheadptr->nextsubareaptr = NULL;
      if(verbose == TRUE)
        printf("\nAdding first node %d,%d and subarea %d to visit list",
                                          xpos,ypos,currentsubarea);
    }
    else  /* there exist subareas in linked list */
    {
      subareaofinterest = insert_subarea(currentsubarea);
      add_to_node_list(subareaofinterest,xpos,ypos);
    }

  }

  /*************************************************************************/
  /*INSERT_SUBAREA..Inserts subarea in proper place in subarea list and    */
  /*returns apointer to that subarea                                       */
  /*************************************************************************/
  SUBAREALINK insert_subarea(currentsubarea)

    int currentsubarea;

  {

    SUBAREALINK currsubareaptr,prevsubareaptr; /* current and previous pointers */
    SUBAREALINK tempptr;

    currsubareaptr = zh;
    prevsubareaptr = currsubareaptr;

    /* cycle through subareas to place prioritized subarea of interest */
    while (currsubareaptr != NULL)
    {
      /* equivalent subarea priority */
      if (currsubareaptr->subareanum == currentsubarea)
      {
        return currsubareaptr;
      }
      /* advance currentsubarea pointer */
      currsubareaptr = currsubareaptr->nextsubareaptr;
      /* highest priority subarea in list */
      if (zh->subareanum > currentsubarea && currsubareaptr == NULL)
      {
        tempptr=(Activesubarea*)malloc(sizeof(Activesubarea));
        tempptr->subareanum = currentsubarea;
        tempptr->nextsubareaptr = zh;
        tempptr->nodelistptr = NULL;
        zh = tempptr;
        if(verbose == TRUE)
          printf("\nAdding subarea %d to visit list",currentsubarea);
        return tempptr;
      }
      /* lowest priority subarea in list */
      if (prevsubareaptr->subareanum < currentsubarea && currsubareaptr == NULL)
      {
        tempptr=(Activesubarea*)malloc(sizeof(Activesubarea));
        tempptr->subareanum = currentsubarea;
        tempptr->nextsubareaptr = currsubareaptr;
        tempptr->nodelistptr = NULL;
        prevsubareaptr->nextsubareaptr = tempptr;
        if(verbose == TRUE)
          printf("\nAdding subarea %d to visit list",currentsubarea);
        return tempptr;
      }

      /* not highest or lowest priority subarea in list */
      if (prevsubareaptr->subareanum < currentsubarea &&
          currsubareaptr->subareanum > currentsubarea)
      {
        tempptr=(Activesubarea*)malloc(sizeof(Activesubarea));
        tempptr->subareanum = currentsubarea;
        tempptr->nextsubareaptr = currsubareaptr;
        tempptr->nodelistptr = NULL;
        prevsubareaptr->nextsubareaptr = tempptr;
        if(verbose == TRUE)
          printf("\nAdding subarea %d to visit list",currentsubarea);
        return tempptr;
      }
      /* priority location not found, advance pointers for while loop */
      prevsubareaptr = currsubareaptr;

    }

  }

  /*************************************************************************/
  /*ADD_TO_NODE_LIST..Places a node at the head of a node list            */
  /*************************************************************************/
  void add_to_node_list(subareaofint,xposit,yposit)

    SUBAREALINK subareaofint;
    int xposit,yposit;

  {

    NODELINK temptr;  /* temp ptr to new node */

    temptr=(Activenode*)malloc(sizeof(Activenode));
    temptr->x = xposit;
    temptr->y = yposit;
    temptr->nextnodeptr = subareaofint->nodelistptr;
    subareaofint->nodelistptr = temptr;
    if(verbose == TRUE)
      printf("\nAdding node %d,%d to visit list",xposit,yposit);

  }

  /*************************************************************************/
  /*DELETE_FROM_VISIT_LIST..Deletes a node and subarea from visit list     */
  /*************************************************************************/
  void delete_from_visit_list(xpos,ypos)

    int xpos,ypos;

  {

    SUBAREALINK subareaheadptr = NULL;    /* temp ptr to first subarea */
    SUBAREALINK currsubareaptr = NULL;    /* temp ptr to current subarea */
    SUBAREALINK subareaofinterest = NULL; /* ptr to subarea of interest */
  int currentsubarea;                /* indicates subarea of interest number */
  int emptysubarea = FALSE;          /* flag indicating empty subarea */

  currentsubarea = Wld[xpos][ypos].subarea; /* subarea of interest */
  if (zh != NULL)
  {
    subareaofinterest = locate_subarea(currentsubarea);
    emptysubarea = delete_from_node_list(subareaofinterest,xpos,ypos);
    if (emptysubarea == TRUE)
    {
      delete_from_subarea_list(currentsubarea);
    }
  }
  else
  {
    printf("\nERROR! Visit list is empty! \n");
  }

}

/***************************************************************************/
/*LOCATE_SUBAREA..Points to the subarea of interest                        */
/***************************************************************************/
SUBAREALINK locate_subarea(currentsubarea)

  int currentsubarea;

{
  SUBAREALINK tempptr;

  tempptr = zh;
  while (tempptr != NULL)
  {
    if (tempptr->subareanum == currentsubarea)
    {
      if(verbose == TRUE)
        printf("\nSubarea of interest is subarea %d",tempptr->subareanum);
      return tempptr;
    }
    tempptr = tempptr->nextsubareaptr;
  }

}

/***************************************************************************/
/*DELETE_FROM_NODE_LIST..Deletes a node from a list                        */
/***************************************************************************/
int delete_from_node_list(subarea_of_interest,xpos,ypos)

  SUBAREALINK subarea_of_interest;
  int xpos,ypos;

{

  NODELINK tempptr;     /* temp ptr to new node */
  NODELINK previousptr; /* temp ptr to previous node */

  tempptr = subarea_of_interest->nodelistptr;
  /* if node is first in subarea's node list */
  if (tempptr->x == xpos && tempptr->y == ypos)
  {
    previousptr = tempptr;
    subarea_of_interest->nodelistptr = tempptr->nextnodeptr;
    tempptr->nextnodeptr = NULL;
    free(tempptr);
    if(verbose == TRUE)
      printf("\nDeleting node %d,%d from visit list",xpos,ypos);
    if (subarea_of_interest->nodelistptr == NULL) /* no more nodes in list */
```

```
      {
        return TRUE;
      }
    }
    /* if node is not first in subarea of interest */
    /* advance ptr into node list to second node */
    tempptr = tempptr->nextnodeptr;
    previousptr = subarea_of_interest->nodelistptr;

    while (tempptr != NULL)
    {
      if (tempptr->x == xpos && tempptr->y == ypos)
      {
        previousptr->nextnodeptr = tempptr->nextnodeptr;
        tempptr->nextnodeptr = NULL;
        free(tempptr);
        if(verbose == TRUE)
          printf("\nDeleting node %d,%d from visit list",xpos,ypos);
      }
      previousptr = previousptr->nextnodeptr;
      tempptr = tempptr->nextnodeptr;
    }

  }

  /**************************************************************************/
  /*DELETE_FROM_SUBAREA_LIST..Deletes a subarea from a list                */
  /**************************************************************************/
  void delete_from_subarea_list(currentsubarea)

    int currentsubarea;

  {

    SUBAREALINK prevsubarea,currsubarea;

    currsubarea = zh;
    /* deleting first subarea in list */
    if (currsubarea->subareanum == currentsubarea)
    {
      zh = currsubarea->nextsubareaptr;
      currsubarea->nextsubareaptr = NULL;
      free(currsubarea);
      if(verbose == TRUE)
        printf("\nDeleting subarea %d from visit list",currentsubarea);
    }
    /* deleting some other subarea in list */
    prevsubarea = currsubarea;
    currsubarea = currsubarea->nextsubareaptr;
    while (currsubarea != NULL)
    {
      if (currsubarea->subareanum == currentsubarea)
      {
        prevsubarea->nextsubareaptr = currsubarea->nextsubareaptr;
        currsubarea->nextsubareaptr = NULL;
        free(currsubarea);
        if(verbose == TRUE)
          printf("\nDeleting subarea %d from visit list",currentsubarea);
      }
      currsubarea = currsubarea->nextsubareaptr;
      prevsubarea = prevsubarea->nextsubareaptr;
    }

  }

  /**************************************************************************/
  /*GET_PRIORITY_NODE_FROM_VISIT_LIST..Locates highest priority node in    */
  /*visit list based on linear distance then location heuristics          */
```

```
/***************************************************************************/
NODELINK get_priority_node_from_visit_list()

{
  float best_dist = 1000.0,      /* best overall distance at given point */
        dist;                    /* distance to node being analyzed */
  NODELINK best_node,            /* points to best priority node */
           tempptr;              /* points to node being analyzed */

  /* search complete */
  if (zh == NULL)
  {
    return NULL;
  }
  tempptr = zh->nodelistptr;
  while (tempptr != NULL)
  {
    dist = compute_dist(Auv.grid.xpos,tempptr->x,
                        Auv.grid.ypos,tempptr->y);
    if (dist < best_dist)
    {
      best_dist = dist;
      best_node = tempptr;
    }
    if (dist == best_dist)
    {
      best_node = determine_best_direction(best_node,tempptr);
    }
    tempptr = tempptr->nextnodeptr;
  }
  if(verbose == TRUE)
    printf("\nPriority node pointed at in visit list is %d,%d",
                    best_node->x,best_node->y);
  return best_node;

}

/***************************************************************************/
/*DETERMINE_BEST_DIRECTION..Determines which node to go to among nodes    */
/*of equal distance.                                                       */
/***************************************************************************/
NODELINK  determine_best_direction(bestnode,testnode)

  NODELINK bestnode, testnode;

{

  if (bestnode->y > Auv.grid.ypos && testnode->y == Auv.grid.ypos ||
      bestnode->y < Auv.grid.ypos && testnode->y == Auv.grid.ypos)
  {
    return testnode;
  }
  if (bestnode->y == Auv.grid.ypos && testnode->y > Auv.grid.ypos ||
      bestnode->y == Auv.grid.ypos && testnode->y < Auv.grid.ypos)
  {
    return bestnode;
  }
  if (bestnode->y > Auv.grid.ypos  && testnode->y > Auv.grid.ypos ||
      bestnode->y > Auv.grid.ypos  && testnode->y < Auv.grid.ypos ||
      bestnode->y == Auv.grid.ypos && testnode->y == Auv.grid.ypos||
      bestnode->y < Auv.grid.ypos  && testnode->y > Auv.grid.ypos ||
      bestnode->y < Auv.grid.ypos  && testnode->y < Auv.grid.ypos)
  {
    if (bestnode->x > testnode->x)
    {
      if(verbose == TRUE)
        printf("\nBest direction is to node %d,%d",bestnode->x,bestnode->y);
      return bestnode;
    }
    if (bestnode->x < testnode->x || bestnode->x == testnode->x)
    {
      if(verbose == TRUE)
        printf("\nBest direction is to node %d,%d",testnode->x,testnode->y);
      return testnode;
    }
  }

}

/***************************************************************************/
/*ADVANCE_VEHICLE_AND_DELETE_PATH..Given a pathlist, advances vehicle     */
/*along path and deletes path. If obstacle encountered, stops and          */
/*reinitiates new path search                                               */
/***************************************************************************/
void advance_vehicle_and_delete_path(pathlist)

  NEWPATHLINK pathlist;

{

  float tempdist;
  NEWPATHLINK temppathptr;

  while(pathlist != NULL)
  {
    /* compute cumulative distance traveled */
    tempdist = compute_dist(Auv.grid.xpos,pathlist->xpath,
                            Auv.grid.ypos,pathlist->ypath);
    distrav = distrav + tempdist;
    /* compute steps taken by vehicle */
    auvsteps = auvsteps + 1;

    /*clean up old AUV position */
    if (verbose == TRUE)  printf("\n  Clean up old AUV position");
    Wld[Auv.grid.xpos][Auv.grid.ypos].state = VISITED;
    Wld[Auv.grid.xpos][Auv.grid.ypos].visited = TRUE;
    Map[Auv.grid.xpos][Auv.grid.ypos].condition == NOTEVAL;
    /* send updated node data from former AUV position to file */
    /* $ represents vehicle data */
    if (storemissiondata == TRUE)
    {
    fprintf(missionofp,"%c %d %d %d %d %f %d %f %f\n",'$',
                        Wld[Auv.grid.xpos][Auv.grid.ypos].grid.xpos,
                        Wld[Auv.grid.xpos][Auv.grid.ypos].grid.ypos,
                        Wld[Auv.grid.xpos][Auv.grid.ypos].dir,
                        Wld[Auv.grid.xpos][Auv.grid.ypos].state,
                        distrav,auvsteps,0.0,0.0);
    }
    /* update AUV position and status */
    if (verbose == TRUE) printf("\n  Update AUV position");
    if (verbose == TRUE) printf("\n  Pathlist position is %d,%d",
                            pathlist->xpath,pathlist->ypath);
    Auv.grid.xpos = pathlist->xpath;
    Auv.grid.ypos = pathlist->ypath;
    Auv.x = pathlist->xpath;
    Auv.y = pathlist->ypath;
    Wld[pathlist->xpath][pathlist->ypath].state = AUV;
    delete_from_visit_list(pathlist->xpath,pathlist->ypath);
    if(verbose == TRUE)
     printf("\nAUV position = %d,%d\n",
                        Wld[pathlist->xpath][pathlist->ypath].grid.xpos,
                        Wld[pathlist->xpath][pathlist->ypath].grid.ypos);
    /* send updated AUV information to file */
    if(storemissiondata == TRUE)
    /* $ represents vehicle data */
    fprintf(missionofp,"%c %d %d %d %d %f %d %f %f\n",'$',
                        Wld[pathlist->xpath][pathlist->ypath].grid.xpos,
                        Wld[pathlist->xpath][pathlist->ypath].grid.ypos,
                        Wld[pathlist->xpath][pathlist->ypath].dir,
                        Wld[pathlist->xpath][pathlist->ypath].state,
                        distrav,auvsteps,0.0,0.0);

    temppathptr = pathlist;
    pathlist = pathlist->nextnewnode;
    temppathptr->nextnewnode = NULL;
    free(temppathptr);
  }

}

/**************************************************************************/
/*SEARCH_A_STAR..This routine performs the optimal path                  */
/*A-Star search and returns a pointer to path list                       */
/**************************************************************************/
NEWPATHLINK search_a_star(goalnodeptr)

  NODELINK goalnodeptr;

{

  ASNODELINK currentasnodeptr;  /* ptr to asnode of interest */
  ASNODELINK previousasnodeptr; /* ptr to previous asnode of interest */
  NEWPATHLINK pathlistptr;      /* ptr to desired path list */
  Asnode auvnode;   /* temporary holder for auv evaluation posit */
  int i,j;          /* for initializing vertex world */

  /* initialize vertex world */
  for (j=0; j<MAXY; ++j)
  {
    for (i=0; i<MAXX; ++i)
    {
      Map[i][j].xprev = 0;
      Map[i][j].yprev = 0;
      Map[i][j].cumvertcost = 0.0;
      Map[i][j].condition = NOTEVAL;
      Map[i][j].ptrtoagenda = NULL;
      /* works so commented out to save time */
      /* if(verbose == TRUE)
        printf("\n  Vertex node being initialized is %d,%d\n",i,j); */
    }
  }
  /* initialize AUV start position */
  /* makes sure not reanalyzed in algorithm */
  Map[Auv.grid.xpos][Auv.grid.ypos].condition = CHECKED;
  /* beginning with AUV current position ... */
  currentasnodeptr = (Asnode*)malloc(sizeof(Asnode));
  currentasnodeptr->x = Auv.grid.xpos;
  currentasnodeptr->y = Auv.grid.ypos;
  currentasnodeptr->cumcost = 0.0;
  currentasnodeptr->criteria = 0.0;
  currentasnodeptr->nextasnodeptr = NULL;
  /* assign pointer to previous asnode */
  previousasnodeptr = currentasnodeptr;
  if(verbose == TRUE)
    printf("\n  AUV vertex node being initialized is at %d,%d",
                                  Auv.grid.xpos,Auv.grid.ypos);
      /* compute costs and criteria and add to agenda as appropriate */
      /* note, if agenda is empty, use AUV position */
  analyze_neighbor_nodes_a_star(currentasnodeptr,goalnodeptr);
  print_agenda_list();
      /* while agenda not empty, move toward goal */
  while (ash != NULL)
  {
    /* point to principle vertex and delete it from agenda */
    currentasnodeptr = get_principle_node_and_delete_from_agenda();
    /* check for goal */
    if (currentasnodeptr->x == goalnodeptr->x &&
        currentasnodeptr->y == goalnodeptr->y)
    {
      delete_agenda_list(); /* cleans up old agenda */
      pathlistptr = create_path_list(currentasnodeptr);/* builds path list */
      free(currentasnodeptr);
      return pathlistptr;
    }
    /* compute costs and criteria and add to agenda as appropriate */
    previousasnodeptr = currentasnodeptr;
    analyze_neighbor_nodes_a_star(currentasnodeptr,goalnodeptr);
  }

}

/***********************************************************************/
/*ANALYZE_NEIGHBOR_NODES_A_STAR..Looks at neighbor nodes to determine    */
/*status, computes costs and criteria and adds nodes to agenda          */
/***********************************************************************/
void analyze_neighbor_nodes_a_star(currasnodeptr,goalptr)

  ASNODELINK currasnodeptr;
  NODELINK goalptr;

{

  /* NORTH */
  /* check neighbor node */
    /* within confines of world */
  if(inside_area(currasnodeptr->x,currasnodeptr->y+1) &&
    /* and not an obstacle */
     (Wld[currasnodeptr->x][currasnodeptr->y+1].state != OBSTACLE) &&
    /* and either visited or is the goal or is neighbor to the AUV */
    ((Wld[currasnodeptr->x][currasnodeptr->y+1].visited == TRUE) ||
     (currasnodeptr->x == goalptr->x && currasnodeptr->y+1 == goalptr->y) ||
     (adjacent_node(Auv.grid.xpos,Auv.grid.ypos,
                    currasnodeptr->x,currasnodeptr->y+1))) &&
    /* and is either on the frontier or has not been evaluated by a-star */
    ((Map[currasnodeptr->x][currasnodeptr->y+1].condition == FRONTIER) ||
     (Map[currasnodeptr->x][currasnodeptr->y+1].condition == NOTEVAL)))
  {
    if(verbose == TRUE)
      printf("\n  Analyzing neighbor node in Vertex World %d,%d",
                          currasnodeptr->x,currasnodeptr->y+1);
    add_to_agenda_list(currasnodeptr->x,currasnodeptr->y,
                    currasnodeptr->x,currasnodeptr->y+1,goalptr);
  }

                                .
                                .
                                .

  /* NORTHWEST */
  /* check neighbor node */
  if(inside_area(currasnodeptr->x-1,currasnodeptr->y+1) &&
     (Wld[currasnodeptr->x-1][currasnodeptr->y+1].state != OBSTACLE) &&
    ((Wld[currasnodeptr->x-1][currasnodeptr->y+1].visited == TRUE) ||
     (currasnodeptr->x-1 == goalptr->x && currasnodeptr->y+1 == goalptr->y) ||
     (adjacent_node(Auv.grid.xpos,Auv.grid.ypos,
                    currasnodeptr->x-1,currasnodeptr->y+1))) &&
    ((Map[currasnodeptr->x-1][currasnodeptr->y+1].condition == FRONTIER) ||
     (Map[currasnodeptr->x-1][currasnodeptr->y+1].condition == NOTEVAL)))
  {
    if(verbose == TRUE)
      printf("\n  Analyzing neighbor node in Vertex World %d,%d",
                          currasnodeptr->x-1,currasnodeptr->y+1);
    add_to_agenda_list(currasnodeptr->x,currasnodeptr->y,
                        currasnodeptr->x-1,currasnodeptr->y+1,goalptr);
  }

}

/****************************************************************************/
/*ADD_TO_AGENDA_LIST..Adds a vertex node to the agenda list ordered by     */
/*criteria value from lowest to highest                                    */
/****************************************************************************/
void add_to_agenda_list(curx,cury,nextx,nexty,goalasptr)

  int curx,cury,            /* position adding from */
      nextx,nexty;          /* position being added */
  NODELINK goalasptr;       /* points to the goal */

{

  ASNODELINK tempnxtptr,    /* temp ptr to next node being analyzed */
             tempcurptr;    /* temp ptr to current reference node */
  ASNODELINK curragendaptr, /* current agenda location */
             prevagendaptr; /* keeps track of previous agenda location */

  curragendaptr = ash;  /* assign current pointer to head of agenda */
  prevagendaptr = curragendaptr;
  /* allocate memory for node being analyzed */
  tempnxtptr = (Asnode*)malloc(sizeof(Asnode));
  tempnxtptr->x = nextx;
  tempnxtptr->y = nexty;
  /* allocate memory for reference node */
  tempcurptr = (Asnode*)malloc(sizeof(Asnode));
  tempcurptr->x = curx;
  tempcurptr->y = cury;
  if (verbose == TRUE)
    printf("\n  Reference cum cost is %f",Map[curx][cury].cumvertcost);
  /* compute cumulative cost and criteria value */
  /* compute the cumulative cost to next node */
  tempnxtptr->cumcost = Map[curx][cury].cumvertcost +
                        compute_dist(curx,nextx,
                                     cury,nexty);
  if (verbose == TRUE)
    printf("\n  Compute distance between NEXT VERTEX and GOAL -criteria-");
  /* compute criteria value as sum of cumcost and distance to goal */
  tempnxtptr->criteria = tempnxtptr->cumcost +
                         compute_dist(nextx,goalasptr->x,
                                      nexty,goalasptr->y);
  if (verbose == TRUE)
    printf("\n  Criteria is %f",tempnxtptr->criteria);
  tempnxtptr->nextasnodeptr = NULL;
  if (verbose == TRUE)
  {
     printf("\n  Compute distance between");
     printf(" CURRENT VERTEX and NEXT VERTEX -cumcost-");
  }
  /* if a frontier node, check to see if criteria is less */
  /* note that a frontier node is currently on the agenda */
  /* if next node criteria value is less, get rid of old one and */
  /* add next node to agenda list */
  /* else, clean up and do nothing */
  if (Map[tempnxtptr->x][tempnxtptr->y].condition == FRONTIER)
  {
    if (verbose == TRUE)
    {
      printf("\n  Analyzing Frontier Node %d,%d",nextx,nexty);
      printf("\n  Current Criteria is %f",
                  Map[tempnxtptr->x][tempnxtptr->y].ptrtoagenda->criteria);
      printf("\n  Next Criteria is %f",tempnxtptr->criteria);
    }
    if (tempnxtptr->criteria <
        Map[tempnxtptr->x][tempnxtptr->y].ptrtoagenda->criteria)
    {
      if (verbose == TRUE)
        printf("\n  Replacing frontier node %d,%d",
                     tempnxtptr->x,tempnxtptr->y);
      delete_agenda_node(Map[tempnxtptr->x][tempnxtptr->y].ptrtoagenda);
      Map[tempnxtptr->x][tempnxtptr->y].ptrtoagenda = tempnxtptr;
      Map[tempnxtptr->x][tempnxtptr->y].condition = FRONTIER;
      Map[tempnxtptr->x][tempnxtptr->y].cumvertcost = tempnxtptr->cumcost;
    }
    else
    {
      free(tempnxtptr);
      if (verbose == TRUE)
        printf("\n  Not Replacing frontier node %d,%d",
                     tempnxtptr->x,tempnxtptr->y);
      return;
    }
  }
  /* if node is not even on the agenda then change its state and add */
  /* it to agenda list */
  if (Map[tempnxtptr->x][tempnxtptr->y].condition == NOTEVAL)
  {
    Map[tempnxtptr->x][tempnxtptr->y].condition = FRONTIER;
    Map[tempnxtptr->x][tempnxtptr->y].cumvertcost = tempnxtptr->cumcost;
    if (verbose == TRUE)
      printf("\n  Changing Node %d,%d to FRONTIER",
                   tempnxtptr->x,tempnxtptr->y);
  }
  /* add to list ordered by criteria value */
  if (ash != NULL)
  {
    if (verbose == TRUE)
      printf("\n  Adding node %d,%d to agenda",tempnxtptr->x,tempnxtptr->y);
    /* cycle through nodes in list */
    while (curragendaptr != NULL)
    {
      /* lowest criteria value in list */
      if (tempnxtptr->criteria < ash->criteria)
      {
        /* annotate previous node array */
        mark_vertex_path(tempcurptr,tempnxtptr);
        if(verbose == TRUE)
          printf("\n  Inserting vertex %d,%d at head of agenda list",
                     tempnxtptr->x,tempnxtptr->y);
        tempnxtptr->nextasnodeptr = ash;
        ash = tempnxtptr;
        /* clean up pointers */
        free(tempcurptr);
        tempnxtptr = NULL;
        tempcurptr = NULL;
        return;
      }
      /* advance pointer to second node on agenda */
      curragendaptr = curragendaptr->nextasnodeptr;
      /* highest criteria value in list */
      /* not lowest criteria value in list */
      if (curragendaptr == NULL ||
          (tempnxtptr->criteria <= curragendaptr->criteria &&
           tempnxtptr->criteria >= prevagendaptr->criteria))
      {
        /* annotate previous node array */
        mark_vertex_path(tempcurptr,tempnxtptr);
        if(verbose == TRUE)
          printf("\n  Inserting vertex %d,%d within agenda list",
                     tempnxtptr->x,tempnxtptr->y);
          tempnxtptr->nextasnodeptr = curragendaptr;
          prevagendaptr->nextasnodeptr = tempnxtptr;
          /* clean up pointers */
          free(tempcurptr);
          tempnxtptr = NULL;
          tempcurptr = NULL;
          return;
        }
        prevagendaptr = prevagendaptr->nextasnodeptr;
      }
    }
    /* agenda list is empty */
    if (ash == NULL)
    {
      mark_vertex_path(tempcurptr,tempnxtptr);
      if(verbose == TRUE)
        printf("\n  Agenda list is empty. Inserting %d,%d ",
                      tempnxtptr->x,tempnxtptr->y);
      ash = tempnxtptr; /* assign head ptr to first node in list */
      /* clean up pointers */
      free(tempcurptr);
      tempnxtptr = NULL;
      tempcurptr = NULL;
     }

}

/***************************************************************************/
/*DELETE_AGENDA_NODE..Locates node in agenda list and deletes             */
/***************************************************************************/
void delete_agenda_node(agendapointer)

  ASNODELINK agendapointer;

{

  ASNODELINK curptr,prevptr;

  curptr = ash;     /* point to top of list */
  prevptr = curptr;
  /* if first node on agenda list */
  if (ash == agendapointer)
  {
    ash = ash->nextasnodeptr;
    free(curptr);
    return;
  }
  /* not first node on agenda list */
  while (curptr != NULL || curptr != agendapointer)
  {
    curptr = curptr->nextasnodeptr;
    if (curptr == agendapointer)
    {
      prevptr->nextasnodeptr = curptr->nextasnodeptr;
      free(curptr);
      return;
    }
    prevptr = curptr;
  }
  return;

}

/***************************************************************************/
/*GET_PRINCIPLE_NODE_AND_DELETE_FROM_AGENDA..Points to the principle      */
/*node in agenda, returns it, and deletes it from the agenda              */
/***************************************************************************/
ASNODELINK get_principle_node_and_delete_from_agenda()
```

```
{

  ASNODELINK principlenodeptr;

  if(ash == NULL)
    printf("\n  Agenda is empty, cannot get principle node");
  principlenodeptr = ash;
  ash = principlenodeptr->nextasnodeptr;
  principlenodeptr->nextasnodeptr = NULL;
  Map[principlenodeptr->x][principlenodeptr->y].condition = CHECKED;
  if(verbose == TRUE)
    printf("\n  Getting principle node %d,%d from agenda list",
           principlenodeptr->x,principlenodeptr->y);
  return principlenodeptr;

}

/*************************************************************************/
/*DELETE_AGENDA_LIST..Deletes entire agenda list to clean up             */
/*************************************************************************/
void delete_agenda_list()

{

  ASNODELINK tempptr;

  if(verbose == TRUE)
    printf("\n  Goal reached, deleting remaining agenda list");
  while(ash != NULL)
  {
    tempptr = ash;
    ash = ash->nextasnodeptr;
    tempptr->nextasnodeptr = NULL;
    if(verbose == TRUE) printf("\n  Deleting vertex %d,%d from agenda list",
                                 tempptr->x,tempptr->y);
    free(tempptr);
  }

}

/*************************************************************************/
/*MARK_VERTEX_PATH..Annotates previous node in as array node             */
/*************************************************************************/
void mark_vertex_path(prevasnode,currentasnode)

  ASNODELINK prevasnode,     /* where the current node came from */
             currentasnode;  /* current node */

{

  if(verbose == TRUE)
    printf("\n  Vertex array position %d,%d.prev gets %d,%d",
          currentasnode->x,currentasnode->y,prevasnode->x,prevasnode->y);
  /* mark where the current node came from */
  Map[currentasnode->x][currentasnode->y].xprev = prevasnode->x;
  Map[currentasnode->x][currentasnode->y].yprev = prevasnode->y;
  /* points to current node on the agenda */
  Map[currentasnode->x][currentasnode->y].ptrtoagenda = currentasnode;

}

/*************************************************************************/
/*CREATE_PATH_LIST..Creates and builds a path to goal list               */
/*head of list is first node after vehicle, last node is goal            */
/*************************************************************************/
NEWPATHLINK create_path_list(goalptr)
```

```
  ASNODELINK goalptr;

{

  NEWPATHLINK pthd,newptr;

  if(verbose == TRUE)
    printf("\n  Goal reached, creating path list as follows:");

  /* put goal position in list */
  newptr = (Newpath*)malloc(sizeof(Newpath));
  newptr->xpath = goalptr->x;
  newptr->ypath = goalptr->y;
  newptr->nextnewnode = NULL;
  if (storemissiondata == TRUE)
    fprintf(missionofp,"%c %d %d %d %d %f %d %f %f\n",'$',
                             newptr->xpath,
                             newptr->ypath,
                             0,
                             6,
                             distrav,auvsteps,0.0,0.0);
  pthd = newptr;
  if(verbose == TRUE)
    printf("\n  %d,%d",newptr->xpath,newptr->ypath);
  while (((Map[newptr->xpath][newptr->ypath].xprev != Auv.grid.xpos) ||
          (Map[newptr->xpath][newptr->ypath].yprev != Auv.grid.ypos)))
  {
    newptr = (Newpath*)malloc(sizeof(Newpath));
    newptr->xpath = Map[pthd->xpath][pthd->ypath].xprev;
    newptr->ypath = Map[pthd->xpath][pthd->ypath].yprev;
    newptr->nextnewnode = pthd;
    if (storemissiondata == TRUE)
      fprintf(missionofp,"%c %d %d %d %d %f %d %f %f\n",'$',
                             newptr->xpath,
                             newptr->ypath,
                             0,
                             6,
                             distrav,auvsteps,0.0,0.0);
    pthd = newptr;
    if(verbose == TRUE)
      printf("\nPath  %d,%d",newptr->xpath,newptr->ypath);
  }
  return pthd;

}

/**************************************************************************/
/*ODD..Determines if a value is odd or even                               */
/**************************************************************************/
int odd(value)

  int value;
{

  if (value % 2 == 0)
    return EVEN;
  else
    return ODD;

}

/**************************************************************************/
/*COMPUTE_DIST..Determines the distance between two nodes                 */
/**************************************************************************/
float compute_dist(xloc,xpos,yloc,ypos)

  int xloc,xpos,yloc,ypos;
```

```
{

  double dist;

  dist = sqrt((double)(((xloc - xpos)*(xloc - xpos)) +
                    ((yloc - ypos)*(yloc - ypos))));
  if(verbose == TRUE)
    printf("\nThe distance between %d,%d and node %d,%d is %f",
              xloc,yloc,xpos,ypos,(float)dist);
  return (float)dist;
}

/***************************************************************************/
/*INSIDE_AREA..Determines if a node is inside the search area              */
/***************************************************************************/
int inside_area(x,y)

  int x,y;

{

  if (x >= 0 && x < MAXX && y >= 0 && y < MAXY)
    return TRUE;
  else
    return FALSE;

}

/***************************************************************************/
/*IS_ADJACENT..Determine if a node is adjacent to the vehicle            */
/***************************************************************************/
int is_adjacent(a,b,c)

  int a,b,c;  /* node location being analyzed */

{

  int e,f;    /* holder for vehicle position */

  e = Auv.grid.xpos;
  f = Auv.grid.ypos;
  if((a <= e+1 && a >= e-1) &&
      (b <= f+1 && b >= f-1) &&
      (c != AUV))
  {
    return TRUE;
  }
  else
  {
    return FALSE;
  }

}

/***************************************************************************/
/*ADJACENT_NODE..Returns true of one node is adjacent to the other         */
/***************************************************************************/
int adjacent_node(x,y,nextx,nexty)

  int x,y,nextx,nexty;

{

  if((nextx <= x+1 && nextx >= x-1) &&
      (nexty <= y+1 && nexty >= y-1))
    return TRUE;
  else
    return FALSE;
}

/****************************************************************************/
/*CLEAN_UP..Clean up after running program                                  */
/****************************************************************************/
void clean_up()

{

  print_active_list();
  if (storemissiondata == TRUE)
    fclose(missionofp);

}

/****************************************************************************/
/*TEST_ADVANCE..Test routine for advancing AUV through world. Does not     */
/*currently employ a-star or other search.                                  */
/****************************************************************************/
void test_advance(testnode)

  NODELINK testnode;

{

  /* clean up old AUV position */
  Wld[Auv.grid.xpos][Auv.grid.ypos].state = VISITED;
  Wld[Auv.grid.xpos][Auv.grid.ypos].visited = TRUE;
  /* send updated node data from former AUV position to file */
  /* $ represents vehicle data */
  if (storemissiondata == TRUE)
  {
  fprintf(missionofp,"%c %d %d %d %d %f %d %f %f\n",'$',
                              Wld[Auv.grid.xpos][Auv.grid.ypos].grid.xpos,
                              Wld[Auv.grid.xpos][Auv.grid.ypos].grid.ypos,
                              Wld[Auv.grid.xpos][Auv.grid.ypos].dir,
                              Wld[Auv.grid.xpos][Auv.grid.ypos].state,
                              distrav,auvsteps,0.0,0.0);
  }
  /* update AUV position and status */
  Auv.grid.xpos = testnode->x;
  Auv.grid.ypos = testnode->y;
  Auv.x = testnode->x;
  Auv.y = testnode->y;
  Wld[testnode->x][testnode->y].state = AUV;
  if(verbose == TRUE)
    printf("\nAUV position = %d,%d",Wld[testnode->x][testnode->y].grid.xpos,
                                    Wld[testnode->x][testnode->y].grid.ypos);
  /* send updated AUV information to file */
  if(storemissiondata == TRUE)
  /* $ represents vehicle data */
  fprintf(missionofp,"%c %d %d %d %d %f %d %f %f\n",'$',
                                Wld[testnode->x][testnode->y].grid.xpos,
                                Wld[testnode->x][testnode->y].grid.ypos,
                                Wld[testnode->x][testnode->y].dir,
                                Wld[testnode->x][testnode->y].state,
                                distrav,auvsteps,0.0,0.0);
  /* remove visited node from visit list */
  delete_from_visit_list(testnode->x,testnode->y);

}

/****************************************************************************/
/*PRINT_ACTIVE_LIST..Prints contents cf active node list to screen         */
/****************************************************************************/
void print_active_list()
```

```
{

  SUBAREALINK ptrsubarea;
  NODELINK ptrnode;

  ptrsubarea = zh;
  while(ptrsubarea != NULL)
  {
    if(verbose == TRUE) printf("Subarea is %d\n",ptrsubarea->subareanum);
    ptrnode = ptrsubarea->nodelistptr;
    while(ptrnode != NULL)
    {
      if(verbose == TRUE) printf(" Node is %d,%d\n",ptrnode->x,ptrnode->y);
      ptrnode = ptrnode->nextnodeptr;
    }
    ptrsubarea = ptrsubarea->nextsubareaptr;
  }

}

/**************************************************************************/
/*PRINT_AGENDA_LIST..Prints contents of active node list to screen       */
/**************************************************************************/
void print_agenda_list()

{

  ASNODELINK ptrasnode;

  ptrasnode = ash;
  while(ptrasnode != NULL)
  {
     if(verbose == TRUE)  printf("\n    Agenda node is %d,%d",
                              ptrasnode->x,ptrasnode->y);
    ptrasnode = ptrasnode->nextasnodeptr;
  }

}

/**************************************************************************/
/*PRINT_PATH_LIST..Prints contents of path list to screen                */
/**************************************************************************/
void print_path_list(pathptr)

  NEWPATHLINK pathptr;
{

  while(pathptr != NULL)
  {
    if(verbose == TRUE)
       printf("\n    Path node is %d,%d", pathptr->xpath,pathptr->ypath);
    pathptr = pathptr->nextnewnode;
  }

}
```

APPENDIX B: THREE-DIMENSIONAL MINEFIELD SEARCH SOURCE CODE

```
/*************************************************************************
Title:  mine3d.h
Author: Mark Compton
Course: Thesis
Date:   10 Mar 92

Description: This program is a graph-based, three-dimensional search
             algorithm for minefield search missions by AUVs.
Support:     mine3d.c
Display:     Output files may be run in "Search" Graphic simulator.
*************************************************************************/

/*Preprocessing Directives***************************************************/
#define BOXWIDTH  15.0  /* box width              */
#define BOXHEIGHT 15.0  /* box height             */
#define BOXDEPTH  15.0  /* box depth              */
#define MAXX 15         /* number of nodes in x direction 25*/
#define MAXY 15         /* number of nodes in y direction 25*/
#define MAXZ  3         /* number of nodes in z direction 9*/
#define SUBX 3          /* number of subareas in x direction 5*/
#define SUBY 3          /* number of subareas in y direction 5*/
#define SUBZ 3          /* number of subareas in z direction 3*/
#define SUBXNODES 5     /* number of nodes in each subarea in x direction 5*/
#define SUBYNODES 5     /* number of nodes in each subarea in y direction 5*/
#define SUBZNODES 1     /* number of nodes in each subarea in z direction 3*/
#define MAXSTRING 20    /* for file names */

#define ODD  1          /* defines if a value is odd */
#define EVEN 0          /* defines if a value is even */

#define TRUE  1         /* needed on non iris */
#define FALSE 0

/* defines states of a node */
#define FREE      0     /* nothing at node */
#define OBSTACLE 1      /* obstacle at node */
#define AUV       2     /* vehicle (Autonomous Underwater Vehicle) */
#define ADJACENT 3      /* node adjacent to vehicle */
#define ACTIVE    4     /* detected, non-object, not visited */
#define VISITED   5     /* node has been previously visited */
#define ASPATH    6     /* local path selected by A-Star search */

/* defines states of previous node array nodes */
#define NOTEVAL  0      /* node not evaluated - default state */
#define FRONTIER 1      /* node placed on agenda */
#define CHECKED  2      /* node evaluated and taken off agenda */

/* defines direction of vehicle */
#define NODIR 0

/*Globals*******************************************************************/
char outnode[MAXSTRING];       /* output file for nodes */
FILE *obstacleofp;             /* pointer for sending obstacles to file */
FILE *obstacleifp;             /* pointer for receiving obstacles from file */
FILE *missionofp;              /* pointer for sending mission to file */
FILE *missionifp;              /* pointer for receiving mission from file */
FILE *trackofp;                /* pointer for sending track to file */
char inobstacle[MAXSTRING];    /* input file name for obstacles */
char outmission[MAXSTRING];    /* output file name for mission */
char outtrack[MAXSTRING];      /* output file name for track */
```

```
int searchmethod = 0;           /* select search method */
int searchtype = 0;             /* select search type */
int auvstartx = 0;              /* starting position for AUV */
int auvstarty = 0;              /* starting position for AUV */
int auvstartz = 0;              /* starting position for AUV */
int retrievedata = FALSE;       /* flag for retrieving obstacle data */
int verbose = FALSE;            /* flag to control verbose output */
int storemissiondata = FALSE;   /* flag for sending mission data to file */
int storetrackdata = FALSE;     /* flag for sending track data to file */
int apriorіobstacle = FALSE;    /* shows obstacles prior to mission viewing */

/*Structures*****************************************************************/
struct location          /* basic structure for location of an entity */
{
  int xpos,ypos,zpos;    /* grid coordinates */
};

typedef struct location Location;

struct node              /* basic structure of a search area node */
{
  Location grid;         /* grid coordinates of node */
  int subarea;           /* subarea node is located in */
  double x,y,z;          /* world position of node */
  int dir;               /* direction node is looking */
  int state;             /* status of node */
  int visited;           /* node previously visited by vehicle flag */
};
  typedef struct node Node;

Node Wld[MAXX][MAXY][MAXZ]; /* 3-D array for storing search area nodes */

struct vehicle       /* AUV */
{
  Location grid;     /* grid coordinates of Vehicle */
  int subarea;       /* subarea vehicle is located in */
  double x,y,z;      /* world position of vehicle */
  int dir;           /* direction vehicle is looking or traveling */
};
  typedef struct vehicle Vehicle;

Vehicle Auv;           /* Autonomous Underwater Vehicle */

struct activenode;     /* announce future structure */

struct activesubarea   /* structure of subarea in visit list */
{
  int subareanum;                     /* number of the subarea */
  struct activenode *nodelistptr;     /* pointer to first node in list */
  struct activesubarea *nextsubareaptr; /* pointer to next subarea in list */
};
  typedef struct activesubarea Activesubarea;
  typedef Activesubarea *SUBAREALINK;   /* pointer to Activesubarea */
  SUBAREALINK zh;                       /* pointer to first subarea in list */

struct activenode    /* structure of node in visit list */
{
  int x,y,z;                        /* coordinates of node */
  struct activenode *nextnodeptr;   /* pointer to next node in list */
};
  typedef struct activenode Activenode;
  typedef Activenode *NODELINK;     /* pointer to Activenode */

struct asnode    /* structure of node in a_star search agenda list */
{
  int x,y,z;                      /* coordinates of asnode */
  float cumcost;                  /* cumulative cost from start to asnode */
  float criteria;                 /* sum of cumcost and evaluation function */
  struct asnode *nextasnodeptr;  /* points to next asnode in list */
};
  typedef struct asnode Asnode;
  typedef Asnode *ASNODELINK;        /* pointer to Asnode */
  ASNODELINK ash;                    /* points to head of list */

struct asvertex            /* structure for node in previous node array */
{
  int xprev,yprev,zprev;  /* coordinates of previous vertex */
  float cumvertcost;      /* cumulative cost from start */
  int condition;          /* state of the node */
  ASNODELINK ptrtoagenda; /* points to corresponding node in agenda */
};
  typedef struct asvertex Asvertex;
  Asvertex Map[MAXX][MAXY][MAXZ];

struct newpath                   /* structure for ordered path to goal */
{
  int xpath,ypath,zpath;         /* coordinates of nodes in path */
  struct newpath *nextnewnode;   /* pointer to next newpath node */
};
  typedef struct newpath Newpath;
  typedef Newpath *NEWPATHLINK;  /* pointer to Newpath */
  /*NEWPATHLINK pathhead;*/      /* points to head of path list */

/*functions*************************************************************/
/* minefield search functions */
void set_up();
void build_world();
int subarea();
void retrieveobstacledata();
void start_position();
void conduct_mission();
void ladder_search();
void perform_search_routine();
void analyze_adjacent_nodes();
int node_visible();
void add_to_visit_list();
SUBAREALINK insert_subarea();
void add_to_node_list();
void delete_from_visit_list();
SUBAREALINK locate_subarea();
int delete_from_node_list();
void delete_from_subarea_list();
NODELINK get_priority_node_from_visit_list();
NODELINK determine_best_direction();
void advance_vehicle_and_delete_path();

/* A-Star functions */
NEWPATHLINK search_a_star();
void analyze_neighbor_nodes_a_star();
void add_to_agenda_list();
void delete_agenda_node();
ASNODELINK get_principle_node_and_delete_from_agenda();
void delete_agenda_list();
void mark_vertex_path();
NEWPATHLINK create_path_list();

/*General functions */
int odd();
int even();
float compute_dist();
int inside_area();
int is_adjacent();
int adjacent_node();
void clean_up();
void test_advance();
void print_active_list();
void print_agenda_list();
void print_path_list();


/****************************************************************************
Title:  mine3d.c
Author: Mark Compton
Course: Thesis
Date:   10 Mar 92

Description: This program is a graph-based, three-dimensional search
             algorithm for minefield search missions by AUVs.
****************************************************************************/

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "mine3d.h"

main ()

{

  int testrun = TRUE;

  set_up();
  conduct_mission();
  clean_up();

}

/***************************************************************************/
/*SET_UP..Get user inputs                                                  */
/***************************************************************************/
void set_up()

{

  char answer  = 'n';
  char reply   = 'n';
  char respond = 'n';
  char what    = 'n';
  char ack     = 'n';

  /* Ask if verbose output to screen */
  printf("\nDo you wish verbose output to screen?  ");
  scanf("%c",&reply);
  if (reply == 'y' || reply == 'Y')  verbose = TRUE;
  reply = 'n';
  /* Setup for retrieving obstacle data */
  scanf("%c",&answer); /* hack to clear carrage return from buffer */
  printf("\n\nObstacle data may be retrieved from a file. \n\n");
  printf("Will you be retrieving data from a file?  ");
  scanf("%c",&answer);
  if (answer == 'y' || answer == 'Y')
  {
    retrievedata = TRUE;
    printf("\n\nPlease enter the obstacle input file name: \n");
    scanf("%s",inobstacle);
  }
  /* Setup for sending mission data to file */
  scanf("%c",&respond); /* hack to clear carrage return from buffer */
  printf("\nMission data may be saved to a file. \n\n");
  printf("Will you be sending mission data to a file?  ");
  scanf("%c",&respond);
  if (respond == 'y' || respond == 'Y')
  {
    storemissiondata = TRUE;
    printf("\n\nPlease enter the mission file name: \n");
    scanf("%s",outmission);
    missionofp = fopen(outmission,"w");    /* open the mission file */
  }
  /* Setup for sending mission track to file */
  scanf("%c",&what); /* hack to clear carrage return from buffer */
  printf("\nMission track may be saved to a file. \n\n");
  printf("Will you be sending mission track to a file?  ");
  scanf("%c",&what);
  if (what == 'y' || what == 'Y')
  {
    storetrackdata = TRUE;
    printf("\n\nPlease enter the track file name: \n");
    scanf("%s",outtrack);
    trackofp = fopen(outtrack,"w");    /* open the track file */
  }
  /* Setup for showing obstacles prior to viewing mission playback */
  scanf("%c",&ack); /* hack to clear carrage return from buffer */
  printf("\nObstacles may be displayed prior to viewing mission. \n\n");
  printf("Do you wish to view obstacles prior to viewing mission?  ");
  scanf("%c",&ack);
  if (ack == 'y' || ack == 'Y')
  {
    apriorioobstacle = TRUE;
  }
  /* Select search method */
  printf("\nSelect search method from the following file by typing \n");
  printf("the desired number:  ");
  printf("\n\n1. Ladder with Sub-area priorities.");
  printf("\n2. Ladder without Sub-area priorities. ");
  printf("\n3. None.\n");
  scanf("%d",&searchmethod);
  /* Select search type */
  if(searchmethod == 1 || searchmethod == 2)
  {
    printf("\nSelect search type from the following file by typing \n");
    printf("the desired number:  ");
    printf("\n\n1. A Star.");
    printf("\n2. None. ");
    printf("\n3. None. ");
    printf("\n4. Test Advance. \n");
    scanf("%d",&searchtype);
  }
  /* Select AUV starting position */
  printf("\nSelect the start position for the AUV (integer value): \n");
  printf("\nx =  ");
  scanf("%d",&auvstartx);
  printf("\ny =  ");
  scanf("%d",&auvstarty);
  printf("\nz =  ");
  scanf("%d",&auvstartz);
  build_world();  /* initialize array and node structure */

}

/******************************************************************************/
/*BUILD_WORLD..Builds the world for vehicle operations                        */
/******************************************************************************/
void build_world()

{

  int i,j,k;  /* variables for rows, columns and height */

  for(k=0; k < MAXZ; k=k+1)
  {
    for(j=0; j < MAXY; j=j+1)
    {
        for(i=0; i < MAXX; i=i+1)
        {
          Wld[i][j][k].x = i;
          Wld[i][j][k].y = j;
          Wld[i][j][k].z = k;
          Wld[i][j][k].state = FREE;
          Wld[i][j][k].visited = FALSE;
          Wld[i][j][k].grid.xpos = i;
          Wld[i][j][k].grid.ypos = j;
          Wld[i][j][k].grid.zpos = k;
          if (searchmethod == 2)
          {
            Wld[i][j][k].subarea = 1;
          }
          else
          {
            Wld[i][j][k].subarea = subarea(i,j,k);
          }
      /* following commented out due to excessive verbose time consumed */
      /* if(verbose == TRUE)
          printf("\nWorld Node is %d,%d,%d",
                        Wld[i][j][k].grid.xpos,Wld[i][j][k].grid.ypos,
                        Wld[i][j][k].grid.zpos); */
        }
      }
    }
    if (retrievedata == TRUE)
    {
    retrieveobstacledata();    /* fill in obstacles */
    }
    start_position();          /* initiates AUV start position */

}

/**************************************************************************/
/*SUBAREA..Determines which subarea a node is in                          */
/**************************************************************************/
int subarea(a,b,c)

int a,b,c; /* feed in the grid coordinates */

{

  static int i,j,k,row,column,level,subarea;

  for (k = 1; k <= SUBZ; ++k)                  /* step thru subarea levels */
  {
    for (i = 1; i <= SUBY; ++i)                /* step thru subarea rows */
    {
      for (j = 1; j <= SUBX; ++j)              /* step thru subarea columns */
      {
        if (c >= (k-1)*SUBZNODES && c < k*SUBZNODES)
        {
          level = k;                           /* subarea level */
        }
        if (b >= (i-1)*SUBYNODES && b < i*SUBYNODES)
        {
          row = i;                             /* subarea row */
        }
        if (a >= (j-1)*SUBXNODES && a < j*SUBXNODES)
        {
          column = j;                          /* subarea column */
        }
      }
    }
  }
  if (odd(level))  /* increase from right to left in odd rows */
  {
    if (odd(row))
    {
      subarea = (((row-1)*(SUBZ*SUBX))+column)+((level-1)*SUBX);
    }
    if (even(row))
    {
      subarea = (((row*SUBZ*SUBX)+1)-column)-((level-1)*SUBX);
    }
  }
  if (even(level)) /* decrease from right to left in odd rows */
  {
    if (odd(row))
    {
      subarea = (((row*SUBZ*SUBX)+1)-column)-((level-1)*SUBX);
    }
    if (even(row))
    {
      subarea = ((((row-1)*SUBZ*SUBX))+column)+((level-1)*SUBX);
    }
  }
  /* verbose output commented out, too verbose */
  /*if(verbose == TRUE)
    printf("\nSubarea is %d",subarea);
  */
  return subarea;

}

/***************************************************************************/
/*RETRIEVEOBSTACLEDATA..Retrieves obstacle data from file if requested   */
/***************************************************************************/
void retrieveobstacledata()

{

  int x,y,z;

  obstacleifp = fopen(inobstacle,"r");
  /* read obstacle positions from obstacle file */
  while(fscanf(obstacleifp, "%d%d%d", &x,&y,&z) != EOF)
  {
    Wld[x][y][z].state = OBSTACLE;
    if(verbose == TRUE)
      printf("\nObstacle at %d,%d,%d",x,y,z);
    /* send obstacle world data to mission file */
    /* note that % indicates world data */
    if(storemissiondata == TRUE && aprioriobstacle == TRUE)
    fprintf(missionofp,"%c %d %d %d %d %d\n",'%',
                                     Wld[x][y][z].grid.xpos,
                                     Wld[x][y][z].grid.ypos,
                                     Wld[x][y][z].grid.zpos,
                                     Wld[x][y][z].dir,
                                     Wld[x][y][z].state);
  }
  fclose(obstacleifp);

}

/***************************************************************************/
/*START_POSITION..Defines the starting position of the AUV               */
/***************************************************************************/
void start_position()

{

  /* read start position from global variable */
  Auv.x = Wld[auvstartx][auvstarty][auvstartz].x;
  Auv.y = Wld[auvstartx][auvstarty][auvstartz].y;
  Auv.z = Wld[auvstartx][auvstarty][auvstartz].z;
  Auv.grid.xpos = Wld[auvstartx][auvstarty][auvstartz].grid.xpos;
  Auv.grid.ypos = Wld[auvstartx][auvstarty][auvstartz].grid.ypos;
  Auv.grid.zpos = Wld[auvstartx][auvstarty][auvstartz].grid.zpos;
  Auv.dir = NODIR;
  Wld[auvstartx][auvstarty][auvstartz].state = AUV;
  Wld[auvstartx][auvstarty][auvstartz].visited = TRUE;
  if(verbose == TRUE)
    printf("\nAUV start = %d,%d,%d",Auv.grid.xpos,Auv.grid.ypos,Auv.grid.zpos);
  /* send AUV start position to file */
  /* note that $ indicates search path data */
  if (storemissiondata == TRUE)
  {
  /* send start AUV data to file */
  fprintf(missionofp,"%c %d %d %d %d %d\n",'$',
                     Wld[auvstartx][auvstarty][auvstartz].grid.xpos,
                     Wld[auvstartx][auvstarty][auvstartz].grid.ypos,
                     Wld[auvstartx][auvstarty][auvstartz].grid.zpos,
                     Wld[auvstartx][auvstarty][auvstartz].dir,
                     Wld[auvstartx][auvstarty][auvstartz].state);
  /* clean up old AUV position and send to file */
  /* $ represents vehicle data */
  fprintf(missionofp,"%c %d %d %d %d %d\n",'$',
                     Wld[auvstartx][auvstarty][auvstartz].grid.xpos,
                     Wld[auvstartx][auvstarty][auvstartz].grid.ypos,
                     Wld[auvstartx][auvstarty][auvstartz].grid.zpos,
                     NODIR,
                     VISITED);
  }

}

/**************************************************************************/
/*CONDUCT_MISSION..Initiates selected mission                            */
/**************************************************************************/
void conduct_mission()

{

  if(searchmethod == 1)
  {
    ladder_search();
  }
  if(searchmethod == 2)
  {
    ladder_search();
  }
  if(searchmethod == 3)
  {
    if(verbose == TRUE)
    printf("\nThis search method is not yet programmed.\n");
  }

}

/**************************************************************************/
/*LADDER_SEARCH..Conducts modified ladder search                          */
/* Logic as follows:
    1. Look at nodes adjacent to vehicle to determine their status.
       a. If nodes have not been visited and are not obstacles and
             are not currently on the visit list, add them to the
             visit list.
    2. Determine the highest priority node.
       a. Highest priority node is in list of highest priority subarea.
       b. Within highest priority subarea, determine closest node to vehicle.
       c. If more than one closest node, choose horizontal over vertical,
          right over left, up over down.
    3. Move vehicle to priority node.
    4. Delete visited node from visit list.
    5. Search complete when visit list is empty.
*/
/****************************************************************************/
void ladder_search()

{

  NODELINK visitnode = NULL; /* priority node to visit from visit list */

  analyze_adjacent_nodes();
  visitnode = get_priority_node_from_visit_list();
  if(verbose == TRUE)
    printf("\nThe priority node is %d,%d,%d\n",
                    visitnode->x,visitnode->y,visitnode->z);
  while (visitnode != NULL)
  {
    perform_search_routine(visitnode); /* moves AUV to priority node */
    analyze_adjacent_nodes();
    visitnode = get_priority_node_from_visit_list();
    /* if(verbose == TRUE) */
      /* printf("\nThe priority node is %d,%d,%d",
                      visitnode->x,visitnode->y,visitnode->z); */
  }
  if (visitnode == NULL)
  {
    printf("\nSEARCH COMPLETE!!\n");
  }

}

/****************************************************************************/
/*PERFORM_SEARCH_ROUTINE..This selects the type of search routine used to*/
/*go to the selected goal node, conducts search and returns path        */
/****************************************************************************/
void perform_search_routine(goalnodeptr)

  NODELINK goalnodeptr; /* node to be visited from AUV current posit */

{

  NEWPATHLINK pathlistptr; /* points to list giving vehicle path */

  if (searchtype == 1)
  {
  if(verbose == TRUE)
    printf("\n\nBeginning A-star search to next goal node\n\n");
    /* determine path to goal from AUV current position with A-star */
    pathlistptr = search_a_star(goalnodeptr);
    /* move vehicle along path, replan if obstacle encountered */
    /* and delete path list when finished */
    /* print_path_list(pathlistptr); */
    advance_vehicle_and_delete_path(pathlistptr);
    free(pathlistptr);
    if(verbose == TRUE)
      printf("\n\nEnding A-star search ... reached this goal\n\n");
  }
  if (searchtype == 2) printf("\nNot a usable search!!\n");
  if (searchtype == 3) printf("\nNot a usable search!!\n");
  if (searchtype == 4)
  {
    if(verbose == TRUE)
      printf("\nThe search routine is test advance\n");
    test_advance(goalnodeptr);
  }

}
```

```
/****************************************************************************/
/*ANALYZE_ADJACENT_NODES..Determine status of nodes adjacent to AUV       */
/*if nodes is not an obstacle and has not been visited then add it to     */
/*the visit list                                                           */
/****************************************************************************/
void analyze_adjacent_nodes()

{

/* LOOK AT UP */

  /* UP NORTH */
  /* check adjacent node */
  if((node_visible(Auv.grid.xpos,Auv.grid.ypos+1,Auv.grid.zpos+1) == TRUE) &&
     Wld[Auv.grid.xpos][Auv.grid.ypos+1][Auv.grid.zpos+1].state == OBSTACLE &&
     storemissiondata == TRUE)
  {
    fprintf(missionofp,"%c %d %d %d %d %d\n",'%',
      Wld[Auv.grid.xpos][Auv.grid.ypos+1][Auv.grid.zpos+1].grid.xpos,
      Wld[Auv.grid.xpos][Auv.grid.ypos+1][Auv.grid.zpos+1].grid.ypos,
      Wld[Auv.grid.xpos][Auv.grid.ypos+1][Auv.grid.zpos+1].grid.zpos,
      Wld[Auv.grid.xpos][Auv.grid.ypos+1][Auv.grid.zpos+1].dir,
      Wld[Auv.grid.xpos][Auv.grid.ypos+1][Auv.grid.zpos+1].state);
  }
  if((node_visible(Auv.grid.xpos,Auv.grid.ypos+1,Auv.grid.zpos+1) == TRUE) &&
    (Wld[Auv.grid.xpos][Auv.grid.ypos+1][Auv.grid.zpos+1].visited == FALSE) &&
    (Wld[Auv.grid.xpos][Auv.grid.ypos+1][Auv.grid.zpos+1].state != ACTIVE)  &&
    (Wld[Auv.grid.xpos][Auv.grid.ypos+1][Auv.grid.zpos+1].state != OBSTACLE))
  {
    if(verbose == TRUE)
      printf("\nAnalyzing adjacent node %d,%d,%d",
                         Auv.grid.xpos,Auv.grid.ypos+1,Auv.grid.zpos+1);
    add_to_visit_list(Auv.grid.xpos,Auv.grid.ypos+1,Auv.grid.zpos+1);
  }

                              .
                              .
                              .

  /* DOWN NORTHWEST */
  /* check adjacent node */
  if((node_visible(Auv.grid.xpos-1,Auv.grid.ypos+1,Auv.grid.zpos-1) == TRUE) &&
     Wld[Auv.grid.xpos-1][Auv.grid.ypos+1][Auv.grid.zpos-1].state == OBSTACLE &&
     storemissiondata == TRUE)
  {
    fprintf(missionofp,"%c %d %d %d %d %d\n",'%',
      Wld[Auv.grid.xpos-1][Auv.grid.ypos+1][Auv.grid.zpos-1].grid.xpos,
      Wld[Auv.grid.xpos-1][Auv.grid.ypos+1][Auv.grid.zpos-1].grid.ypos,
      Wld[Auv.grid.xpos-1][Auv.grid.ypos+1][Auv.grid.zpos-1].grid.zpos,
      Wld[Auv.grid.xpos-1][Auv.grid.ypos+1][Auv.grid.zpos-1].dir,
      Wld[Auv.grid.xpos-1][Auv.grid.ypos+1][Auv.grid.zpos-1].state);
  }
  if((node_visible(Auv.grid.xpos-1,Auv.grid.ypos+1,Auv.grid.zpos-1) == TRUE) &&
    (Wld[Auv.grid.xpos-1][Auv.grid.ypos+1][Auv.grid.zpos-1].visited == FALSE) &&
    (Wld[Auv.grid.xpos-1][Auv.grid.ypos+1][Auv.grid.zpos-1].state != ACTIVE)  &&
    (Wld[Auv.grid.xpos-1][Auv.grid.ypos+1][Auv.grid.zpos-1].state != OBSTACLE))
  {
    if(verbose == TRUE)
      printf("\nAnalyzing adjacent node %d,%d,%d",
                         Auv.grid.xpos-1,Auv.grid.ypos+1,Auv.grid.zpos-1);
    add_to_visit_list(Auv.grid.xpos-1,Auv.grid.ypos+1,Auv.grid.zpos-1);
  }

/* END DOWN */

}

/***************************************************************************/
```

```
/*NODE_VISIBLE..Determines if a node adjacent to AUV is inside of the    */
/*search area and is available                                            */
/**************************************************************************/
int node_visible(x,y,z)

int x,y,z;

{

  if((inside_area(x,y,z)) && (Wld[x][y][z].state == FREE))
    return TRUE;
  else
    return FALSE;

}

/**************************************************************************/
/*ADD_TO_VISIT_LIST..Adds node to visit list                              */
/**************************************************************************/
void add_to_visit_list(xpos,ypos,zpos)

int xpos,ypos,zpos;

{

  SUBAREALINK subareaheadptr = NULL;    /* temp ptr to first subarea */
  SUBAREALINK currsubareaptr = NULL;    /* temp ptr to current subarea */
  SUBAREALINK subareaofinterest = NULL; /* ptr to subarea of interest */
  int currentsubarea;                /* indicates subarea of interest number */

  currentsubarea = Wld[xpos][ypos][zpos].subarea; /* subarea of interest */
  /* change state of node to add to active */
  Wld[xpos][ypos][zpos].state = ACTIVE;
  /* send active node to output file */
  if (storemissiondata == TRUE)
  {
  fprintf(missionofp,"%c %d %d %d %d %d\n",'$',
                         Wld[xpos][ypos][zpos].grid.xpos,
                         Wld[xpos][ypos][zpos].grid.ypos,
                         Wld[xpos][ypos][zpos].grid.zpos,
                         NODIR,
                         Wld[xpos][ypos][zpos].state);
  }
  if (zh == NULL)  /* subarea list is empty so create subarea and first node */
  {
    subareaheadptr = (Activesubarea*)malloc(sizeof(Activesubarea));
    zh = subareaheadptr;        /* attach start pointer to first subarea record */
    subareaheadptr->subareanum = currentsubarea;
    subareaheadptr->nodelistptr = (Activenode*)malloc(sizeof(Activenode));
    subareaheadptr->nodelistptr->x = xpos;
    subareaheadptr->nodelistptr->y = ypos;
    subareaheadptr->nodelistptr->z = zpos;
    subareaheadptr->nodelistptr->nextnodeptr = NULL;
    subareaheadptr->nextsubareaptr = NULL;
    if(verbose == TRUE)
      printf("\nAdding first node %d,%d,%d and subarea %d to visit list",
                                    xpos,ypos,zpos,currentsubarea);
  }
  else  /* there exist subareas in linked list */
  {
    subareaofinterest = insert_subarea(currentsubarea);
    add_to_node_list(subareaofinterest,xpos,ypos,zpos);
  }

}

/**************************************************************************/
/*INSERT_SUBAREA..Inserts subarea in proper place in subarea list         */
```

```
/*and returns a pointer to that subarea                                    */
/**************************************************************************/
SUBAREALINK insert_subarea(currentsubarea)

  int currentsubarea;

{

  SUBAREALINK currsubareaptr,prevsubareaptr; /* current and previous pointers */
  SUBAREALINK tempptr;

  currsubareaptr = zh;
  prevsubareaptr = currsubareaptr;

  /* cycle through subareas to place prioritized subarea of interest */
  while (currsubareaptr != NULL)
  {
    /* equivalent subarea priority */
    if (currsubareaptr->subareanum == currentsubarea)
    {
      return currsubareaptr;
    }
    /* advance currentsubarea pointer */
    currsubareaptr = currsubareaptr->nextsubareaptr;
    /* highest priority subarea in list */
    if (zh->subareanum > currentsubarea &&
        currsubareaptr == NULL)
    {
      tempptr=(Activesubarea*)malloc(sizeof(Activesubarea));
      tempptr->subareanum = currentsubarea;
      tempptr->nextsubareaptr = zh;
      tempptr->nodelistptr = NULL;
      zh = tempptr;
      if(verbose == TRUE)
        printf("\nAdding subarea %d to visit list",currentsubarea);
      return tempptr;
    }
    /* lowest priority subarea in list */
    if (prevsubareaptr->subareanum < currentsubarea &&
        currsubareaptr == NULL)
    {
      tempptr=(Activesubarea*)malloc(sizeof(Activesubarea));
      tempptr->subareanum = currentsubarea;
      tempptr->nextsubareaptr = currsubareaptr;
      tempptr->nodelistptr = NULL;
      prevsubareaptr->nextsubareaptr = tempptr;
      if(verbose == TRUE)
        printf("\nAdding subarea %d to visit list",currentsubarea);
      return tempptr;
    }
    /* not highest or lowest priority subarea in list */
    if (prevsubareaptr->subareanum < currentsubarea &&
        currsubareaptr->subareanum > currentsubarea)
    {
      tempptr=(Activesubarea*)malloc(sizeof(Activesubarea));
      tempptr->subareanum = currentsubarea;
      tempptr->nextsubareaptr = currsubareaptr;
      tempptr->nodelistptr = NULL;
      prevsubareaptr->nextsubareaptr = tempptr;
      if(verbose == TRUE)
        printf("\nAdding subarea %d to visit list",currentsubarea);
      return tempptr;
    }
    /* priority location not found, advance pointers for while loop */
    prevsubareaptr = currsubareaptr;
  }

}
```

```
/****************************************************************************/
/*ADD_TO_NODE_LIST..Places a node at the head of a node list              */
/****************************************************************************/
void add_to_node_list(subareaofint,xposit,yposit,zposit)

  SUBAREALINK subareaofint;
  int xposit,yposit,zposit;

{

  NODELINK temptr;  /* temp ptr to new node */

  temptr=(Activenode*)malloc(sizeof(Activenode));
  temptr->x = xposit;
  temptr->y = yposit;
  temptr->z = zposit;
  temptr->nextnodeptr = subareaofint->nodelistptr;
  subareaofint->nodelistptr = temptr;
  if(verbose == TRUE)
    printf("\nAdding node %d,%d,%d to visit list",xposit,yposit,zposit);

}

/****************************************************************************/
/*DELETE_FROM_VISIT_LIST..Deletes a node and subarea from visit list      */
/****************************************************************************/
void delete_from_visit_list(xpos,ypos,zpos)

  int xpos,ypos,zpos;

{

  SUBAREALINK subareaheadptr = NULL;     /* temp ptr to first subarea */
  SUBAREALINK currsubareaptr = NULL;     /* temp ptr to current subarea */
  SUBAREALINK subareaofinterest = NULL; /* ptr to subarea of interest */
  int currentsubarea;                  /* indicates subarea of interest number */
  int emptysubarea = FALSE;            /* flag indicating empty subarea */

  currentsubarea = Wld[xpos][ypos][zpos].subarea; /* subarea of interest */
  if (zh != NULL)
  {
    subareaofinterest = locate_subarea(currentsubarea);
    emptysubarea = delete_from_node_list(subareaofinterest,xpos,ypos,zpos);
    if (emptysubarea == TRUE)
    {
      delete_from_subarea_list(currentsubarea);
    }
  }
  else
  {
    printf("\nERROR! Visit list is empty! \n");
  }

}

/****************************************************************************/
/*LOCATE_SUBAREA..Points to the subarea of interest                       */
/****************************************************************************/
SUBAREALINK locate_subarea(currentsubarea)

  int currentsubarea;

{
  SUBAREALINK tempptr;

  tempptr = zh;
  while (tempptr != NULL)
```

```
    {
      if (tempptr->subareanum == currentsubarea)
      {
        if(verbose == TRUE)
          printf("\nSubarea of interest is subarea %d",tempptr->subareanum);
        return tempptr;
      }
      tempptr = tempptr->nextsubareaptr;
    }

}

/**************************************************************************/
/*DELETE_FROM_NODE_LIST..Deletes a node from a list                       */
/**************************************************************************/
int delete_from_node_list(subarea_of_interest,xpos,ypos,zpos)

  SUBAREALINK subarea_of_interest;
  int xpos,ypos,zpos;

{

  NODELINK tempptr;     /* temp ptr to new node */
  NODELINK previousptr; /* temp ptr to previous node */

  tempptr = subarea_of_interest->nodelistptr;
  /* if node is first in subarea's node list */
  if (tempptr->x == xpos &&
      tempptr->y == ypos &&
      tempptr->z == zpos)
  {
    previousptr = tempptr;
    subarea_of_interest->nodelistptr = tempptr->nextnodeptr;
    tempptr->nextnodeptr = NULL;
    free(tempptr);
    if(verbose == TRUE)
      printf("\nDeleting node %d,%d,%d from visit list",xpos,ypos,zpos);
    if (subarea_of_interest->nodelistptr == NULL) /* no more nodes in list */
    {
      return TRUE;
    }
  }
  /* if node is not first in subarea of interest */
  /* advance ptr into node list to second node */
  tempptr = tempptr->nextnodeptr;
  previousptr = subarea_of_interest->nodelistptr;
  while (tempptr != NULL)
  {
    if (tempptr->x == xpos && tempptr->y == ypos && tempptr->z == zpos)
    {
      previousptr->nextnodeptr = tempptr->nextnodeptr;
      tempptr->nextnodeptr = NULL;
      free(tempptr);
      if(verbose == TRUE)
        printf("\nDeleting node %d,%d,%d from visit list",xpos,ypos,zpos);
    }
    previousptr = previousptr->nextnodeptr;
    tempptr = tempptr->nextnodeptr;
  }

}

/**************************************************************************/
/*DELETE_FROM_SUBAREA_LIST..Deletes a subarea from a list                 */
/**************************************************************************/
void delete_from_subarea_list(currentsubarea)

  int currentsubarea;
```

```
{
  SUBAREALINK prevsubarea,currsubarea;

  currsubarea = zh;
  /* deleting first subarea in list */
  if (currsubarea->subareanum == currentsubarea)
  {
  zh = currsubarea->nextsubareaptr;
  currsubarea->nextsubareaptr = NULL;
  free(currsubarea);
  if(verbose == TRUE)
    printf("\nDeleting subarea %d from visit list",currentsubarea);
  }
  /* deleting some other subarea in list */
  prevsubarea = currsubarea;
  currsubarea = currsubarea->nextsubareaptr;
  while (currsubarea != NULL)
  {
    if (currsubarea->subareanum == currentsubarea)
    {
      prevsubarea->nextsubareaptr = currsubarea->nextsubareaptr;
      currsubarea->nextsubareaptr = NULL;
      free(currsubarea);
      if(verbose == TRUE)
        printf("\nDeleting subarea %d from visit list",currentsubarea);
    }
    currsubarea = currsubarea->nextsubareaptr;
    prevsubarea = prevsubarea->nextsubareaptr;
  }

}

/***************************************************************************/
/*GET_PRIORITY_NODE_FROM_VISIT_LIST..Locates highest priority node in    */
/*visit list based on linear distance                                      */
/***************************************************************************/
NODELINK get_priority_node_from_visit_list()

{

  float best_dist = 1000.0,/* best overall distance at given point */
        dist;                /* distance to node being analyzed */
  NODELINK best_node,        /* points to best priority node */
           tempptr;          /* points to node being analyzed */

  /* search complete */
  if (zh == NULL)
  {
    return NULL;
  }
  tempptr = zh->nodelistptr;
  while (tempptr != NULL)
  {
    dist = compute_dist(Auv.grid.xpos,tempptr->x,
                        Auv.grid.ypos,tempptr->y,
                        Auv.grid.zpos,tempptr->z);
    if (dist < best_dist)
    {
      best_dist = dist;
      best_node = tempptr;
    }
    if (dist == best_dist)
    {
      best_node = determine_best_direction(best_node,tempptr);
    }
    tempptr = tempptr->nextnodeptr;
  }
  if(verbose == TRUE)
    printf("\nPriority node pointed at in visit list is %d,%d,%d",
                    best_node->x,best_node->y,best_node->z);
  return best_node;

}

/***********************************************************************/
/*DETERMINE_BEST_DIRECTION..Determines which node to go to among nodes   */
/*of equal distance. Takes horizontal over vertical, right over left.    */
/*Takes current level, then upper level, then lower level.               */
/***********************************************************************/
NODELINK  determine_best_direction(bestnode,testnode)

  NODELINK bestnode, testnode;

{

 if (bestnode->z == testnode->z)
 {
  if (bestnode->y > Auv.grid.ypos && testnode->y == Auv.grid.ypos ||
      bestnode->y < Auv.grid.ypos && testnode->y == Auv.grid.ypos)
  {
    return testnode;
  }
  if (bestnode->y == Auv.grid.ypos && testnode->y > Auv.grid.ypos ||
      bestnode->y == Auv.grid.ypos && testnode->y < Auv.grid.ypos)
  {
    return bestnode;
  }
  if (bestnode->y > Auv.grid.ypos && testnode->y > Auv.grid.ypos ||
      bestnode->y > Auv.grid.ypos && testnode->y < Auv.grid.ypos ||
      bestnode->y == Auv.grid.ypos && testnode->y == Auv.grid.ypos ||
      bestnode->y < Auv.grid.ypos && testnode->y > Auv.grid.ypos ||
      bestnode->y < Auv.grid.ypos && testnode->y < Auv.grid.ypos)
  {
    if (bestnode->x > testnode->x)
    {
      if(verbose == TRUE)
        printf("\nBest direction is to node %d,%d",bestnode->x,bestnode->y);
      return bestnode;
    }
    if (bestnode->x < testnode->x || bestnode->x == testnode->x)
    {
      if(verbose == TRUE)
        printf("\nBest direction is to node %d,%d",testnode->x,testnode->y);
      return testnode;
    }
  }
 }
 if (bestnode->z == Auv.grid.zpos && testnode->z != Auv.grid.zpos)
 {
   return bestnode;
 }
 if (bestnode->z != Auv.grid.zpos && testnode->z == Auv.grid.zpos)
 {
   return testnode;
 }
 if (bestnode->z > testnode->z)
 {
   return testnode;
 }
 else
 {
   return bestnode;
 }
```

```
}

/***************************************************************************/
/*ADVANCE_VEHICLE_AND_DELETE_PATH..Given a pathlist, advances vehicle    */
/*along path and deletes path. If obstacle encountered, stops and        */
/*reinitiates new path search                                            */
/***************************************************************************/
void advance_vehicle_and_delete_path(pathlist)

  NEWPATHLINK pathlist;

{

  NEWPATHLINK temppathptr;

  while(pathlist != NULL)
  {
    /*clean up old AUV position */
    if (verbose == TRUE)  printf("\n  Clean up old AUV position");
    Wld[Auv.grid.xpos][Auv.grid.ypos][Auv.grid.zpos].state = VISITED;
    Wld[Auv.grid.xpos][Auv.grid.ypos][Auv.grid.zpos].visited = TRUE;
    Map[Auv.grid.xpos][Auv.grid.ypos][Auv.grid.zpos].condition = NOTEVAL;
    /* send updated node data from former AUV position to file */
    /* $ represents vehicle data */
    if (storemissiondata == TRUE)
    {
    fprintf(missionofp,"%c %d %d %d %d %d\n",'$',
               Wld[Auv.grid.xpos][Auv.grid.ypos][Auv.grid.zpos].grid.xpos,
               Wld[Auv.grid.xpos][Auv.grid.ypos][Auv.grid.zpos].grid.ypos,
               Wld[Auv.grid.xpos][Auv.grid.ypos][Auv.grid.zpos].grid.zpos,
               Wld[Auv.grid.xpos][Auv.grid.ypos][Auv.grid.zpos].dir,
               Wld[Auv.grid.xpos][Auv.grid.ypos][Auv.grid.zpos].state);
    }

    /* update AUV position and status */
    if (verbose == TRUE) printf("\n  Update AUV position");
    if (verbose == TRUE) printf("\n  Pathlist position is %d,%d,%d",
                         pathlist->xpath,pathlist->ypath,pathlist->zpath);
    Auv.grid.xpos = pathlist->xpath;
    Auv.grid.ypos = pathlist->ypath;
    Auv.grid.zpos = pathlist->zpath;
    Auv.x = pathlist->xpath;
    Auv.y = pathlist->ypath;
    Auv.z = pathlist->zpath;
    Wld[pathlist->xpath][pathlist->ypath][pathlist->zpath].state = AUV;
    delete_from_visit_list(pathlist->xpath,pathlist->ypath,pathlist->zpath);
    if(verbose == TRUE)
     printf("\nAUV position = %d,%d,%d\n",
           Wld[pathlist->xpath][pathlist->ypath][pathlist->zpath].grid.xpos,
           Wld[pathlist->xpath][pathlist->ypath][pathlist->zpath].grid.ypos,
           Wld[pathlist->xpath][pathlist->ypath][pathlist->zpath].grid.zpos);
    /* send updated AUV information to file */
    if(storemissiondata == TRUE)
    /* $ represents vehicle data */
    fprintf(missionofp,"%c %d %d %d %d %d\n",'$',
           Wld[pathlist->xpath][pathlist->ypath][pathlist->zpath].grid.xpos,
           Wld[pathlist->xpath][pathlist->ypath][pathlist->zpath].grid.ypos,
           Wld[pathlist->xpath][pathlist->ypath][pathlist->zpath].grid.zpos,
           Wld[pathlist->xpath][pathlist->ypath][pathlist->zpath].dir,
           Wld[pathlist->xpath][pathlist->ypath][pathlist->zpath].state);
    /* send mission track data to file */
    if(storetrackdata == TRUE)
    fprintf(trackofp,"%f %f %f\n",
           (BOXWIDTH *Wld[pathlist->xpath][pathlist->ypath][pathlist->zpath].x),
           (BOXHEIGHT*Wld[pathlist->xpath][pathlist->ypath][pathlist->zpath].y),
           (BOXDEPTH*Wld[pathlist->xpath][pathlist->ypath][pathlist->zpath].z));

    temppathptr = pathlist;
    pathlist = pathlist->nextnewnode;
    temppathptr->nextnewnode = NULL;
    free(temppathptr);
  }

}

/************************************************************************/
/*SEARCH_A_STAR..This routine performs the high level functions of an    */
/*A-Star search and returns a pointer to path list                       */
/************************************************************************/
NEWPATHLINK search_a_star(goalnodeptr)

  NODELINK goalnodeptr;

{

  ASNODELINK currentasnodeptr;  /* ptr to asnode of interest */
  ASNODELINK previousasnodeptr; /* ptr to previous asnode of interest */
  NEWPATHLINK pathlistptr;      /* ptr to desired path list */
  Asnode auvnode;   /* temporary holder for auv evaluation posit */
  int i,j,k;        /* for initializing vertex world */

  /* initialize vertex world */
  for (k=0; k<MAXZ; ++k)
  {
    for (j=0; j<MAXY; ++j)
    {
      for (i=0; i<MAXX; ++i)
      {
        Map[i][j][k].xprev = 0;
        Map[i][j][k].yprev = 0;
        Map[i][j][k].zprev = 0;
        Map[i][j][k].cumvertcost = 0.0;
        Map[i][j][k].condition = NOTEVAL;
        Map[i][j][k].ptrtoagenda = NULL;
        /* works so commented out to save time */
        /* if(verbose == TRUE)
          printf("\n  Vertex node being initialized is %d,%d,%d\n",i,j,k); */
      }
    }
  }
  /* initialize AUV start position */
  /* makes sure not reanalyzed in algorithm */
  Map[Auv.grid.xpos][Auv.grid.ypos][Auv.grid.zpos].condition = CHECKED;
  /* beginning with AUV current position ... */
  currentasnodeptr = (Asnode*)malloc(sizeof(Asnode));
  currentasnodeptr->x = Auv.grid.xpos;
  currentasnodeptr->y = Auv.grid.ypos;
  currentasnodeptr->z = Auv.grid.zpos;
  currentasnodeptr->cumcost = 0.0;
  currentasnodeptr->criteria = 0.0;
  currentasnodeptr->nextasnodeptr = NULL;
  /* assign pointer to previous asnode */
  previousasnodeptr = currentasnodeptr;
  if(verbose == TRUE)
    printf("\n  AUV vertex node being initialized is at %d,%d,%d",
                        Auv.grid.xpos,Auv.grid.ypos,Auv.grid.zpos);
      /* compute costs and criteria and add to agenda as appropriate */
      /* note, if agenda is empty, use AUV position */
  analyze_neighbor_nodes_a_star(currentasnodeptr,goalnodeptr);
  print_agenda_list();
      /* while agenda not empty, move toward goal */
  while (ash != NULL)
  {
        /* point to priority vertex and delete it from agenda */
    currentasnodeptr = get_principle_node_and_delete_from_agenda();
        /* annotate prior vertex in asvertex array */
    /* mark_vertex_path(previousasnodeptr,currentasnodeptr); delete */
        /* check for goal */
    if (currentasnodeptr->x == goalnodeptr->x &&
        currentasnodeptr->y == goalnodeptr->y &&
        currentasnodeptr->z == goalnodeptr->z)
    {
      delete_agenda_list(); /* cleans up old agenda */
      pathlistptr = create_path_list(currentasnodeptr);/* builds path list */
      free(currentasnodeptr);
      return pathlistptr;
    }
        /* compute costs and criteria and add to agenda as appropriate */
    previousasnodeptr = currentasnodeptr;
    analyze_neighbor_nodes_a_star(currentasnodeptr,goalnodeptr);
  }

}

/**************************************************************************/
/*ANALYZE_NEIGHBOR_NODES_A_STAR..Looks at neighbor nodes to determine    */
/*status, computes costs and criteria and adds nodes to agenda           */
/**************************************************************************/
void analyze_neighbor_nodes_a_star(currasnodeptr,goalptr)

  ASNODELINK currasnodeptr;
  NODELINK goalptr;

{

/* LOOK UP */

  /* UP NORTH */
  if(verbose == TRUE)
    printf("\n  Looking Up North");
  /* check neighbor node */
    /* within confines of world */
 if(inside_area(currasnodeptr->x,currasnodeptr->y+1,currasnodeptr->z+1)       &&
    /* and not an obstacle */
    (Wld[currasnodeptr->x][currasnodeptr->y+1][currasnodeptr->z+1].state !=
                                                                OBSTACLE)     &&
    /* and either visited or is the goal or is neighbor to the AUV */
    ((Wld[currasnodeptr->x][currasnodeptr->y+1][currasnodeptr->z+1].visited ==
                                                                      TRUE)  ||
     (currasnodeptr->x == goalptr->x &&
      currasnodeptr->y+1 == goalptr->y &&
      currasnodeptr->z+1 == goalptr->z)                                      ||
     (adjacent_node(Auv.grid.xpos,Auv.grid.ypos,Auv.grid.zpos,
            currasnodeptr->x,currasnodeptr->y+1,currasnodeptr->z+1)))        &&
   ((Map[currasnodeptr->x][currasnodeptr->y+1][currasnodeptr->z+1].condition
                                                            == FRONTIER) ||
    (Map[currasnodeptr->x][currasnodeptr->y+1][currasnodeptr->z+1].condition
                                                            == NOTEVAL)))
 {
   if(verbose == TRUE)
     printf("\n  Analyzing neighbor node in Vertex World %d,%d,%d",
             currasnodeptr->x,currasnodeptr->y+1,currasnodeptr->z+1);
   add_to_agenda_list(currasnodeptr->x,currasnodeptr->y,currasnodeptr->z,
             currasnodeptr->x,currasnodeptr->y+1,currasnodeptr->z+1,goalptr);
 }

                                  .
                                  .
                                  .

 /* DOWN NORTHWEST */
  if(verbose == TRUE)
    printf("\n  Looking Down Northwest");
 /* check neighbor node */
```

```
if(inside_area(currasnodeptr->x-1,currasnodeptr->y+1,currasnodeptr->z-1)    &&
   (Wld[currasnodeptr->x-1][currasnodeptr->y+1][currasnodeptr->z-1].state !=
                                                              OBSTACLE)     &&
  ((Wld[currasnodeptr->x-1][currasnodeptr->y+1][currasnodeptr->z-1].visited ==
                                                                    TRUE) ||
   (currasnodeptr->x-1 == goalptr->x &&
    currasnodeptr->y+1 == goalptr->y &&
    currasnodeptr->z-1 == goalptr->z) ||
   (adjacent_node(Auv.grid.xpos,Auv.grid.ypos,Auv.grid.zpos,
          currasnodeptr->x-1,currasnodeptr->y+1,currasnodeptr->z-1)))    &&
  ((Map[currasnodeptr->x-1][currasnodeptr->y+1][currasnodeptr->z-1].condition
                                                           == FRONTIER) ||
   (Map[currasnodeptr->x-1][currasnodeptr->y+1][currasnodeptr->z-1].condition
                                                           == NOTEVAL)))
 {
   if(verbose == TRUE)
     printf("\n  Analyzing neighbor node in Vertex World %d,%d,%d",
            currasnodeptr->x-1,currasnodeptr->y+1,currasnodeptr->z-1);
   add_to_agenda_list(currasnodeptr->x,currasnodeptr->y,currasnodeptr->z,
            currasnodeptr->x-1,currasnodeptr->y+1,currasnodeptr->z-1,goalptr);
 }

/* END DOWN */

}

/************************************************************************/
/*ADD_TO_AGENDA_LIST..Adds a vertex node to the agenda list ordered by   */
/*criteria value from lowest to highest                                  */
/************************************************************************/
void add_to_agenda_list(curx,cury,curz,nextx,nexty,nextz,goalasptr)

  int curx,cury,curz,          /* position adding from */
      nextx,nexty,nextz;       /* position being added */
  NODELINK goalasptr;          /* points to the goal */

{

  ASNODELINK tempnxtptr,      /* temp ptr to next node being analyzed */
             tempcurptr;      /* temp ptr to current reference node */
  ASNODELINK curragendaptr,   /* current agenda location */
             prevagendaptr;   /* keeps track of previous agenda location */

    curragendaptr = ash;
    prevagendaptr = curragendaptr;
    tempnxtptr = (Asnode*)malloc(sizeof(Asnode));
    tempnxtptr->x = nextx;
    tempnxtptr->y = nexty;
    tempnxtptr->z = nextz;
    tempcurptr = (Asnode*)malloc(sizeof(Asnode));
    tempcurptr->x = curx;
    tempcurptr->y = cury;
    tempcurptr->z = curz;
  if (verbose == TRUE)
  {
    printf("\nThe nodes being fed to mark_vertex_path are %d,%d,%d and %d,%d,%d",
      tempcurptr->x,tempcurptr->y,tempcurptr->z,
      tempnxtptr->x,tempnxtptr->y,tempnxtptr->z);
    printf("\n  Reference cumulative cost is %f",
      Map[curx][cury][curz].cumvertcost);
    printf("\n  Compute distance between");
    printf(" CURRENT VERTEX and NEXT VERTEX -cumcost-");
  }
  /* compute cumulative cost and criteria value */
  /* compute the cumulative cost to next node */
    tempnxtptr->cumcost = Map[curx][cury][curz].cumvertcost +
                          compute_dist(curx,nextx,
                                       cury,nexty,
                                                  curz,nextz);
      if (verbose == TRUE)
        printf("\n  Compute distance between NEXT VERTEX and GOAL -criteria-");
      tempnxtptr->criteria = tempnxtptr->cumcost +
                              compute_dist(nextx,goalasptr->x,
                                           nexty,goalasptr->y,
                                           nextz,goalasptr->z);
      if (verbose == TRUE'
        printf("\n  Criteria is %f",tempnxtptr->criteria);
      tempnxtptr->nextasnodeptr = NULL;
      if (verbose == TRUE)
      {
        printf("\n  Compute distance between");
        printf("\n CURRENT VERTEX and NEXT VERTEX -cumcost-");
      }
    /* if a frontier node, check to see if criteria is less */
    /* note that a frontier node is currently on the agenda */
    /* if next node criteria value is less, get rid of old one and */
    /* add next node to agenda list */
    /* else, clean up and do nothing */
    if (Map[tempnxtptr->x][tempnxtptr->y][tempnxtptr->z].condition == FRONTIER)
    {
      if (verbose == TRUE)
      {
        printf("\n  Analyzing Frontier Node %d,%d,%d",nextx,nexty,nextz);
        printf("\n  Current Criteria is %f",
          Map[tempnxtptr->x][tempnxtptr->y][tempnxtptr->z].ptrtoagenda->criteria);
        printf("\n  Next Criteria is %f",tempnxtptr->criteria);
      }
      if (tempnxtptr->criteria <
          Map[tempnxtptr->x][tempnxtptr->y][tempnxtptr->z].ptrtoagenda->criteria)
      {
        if (verbose == TRUE)
          printf("\n  Replacing frontier node %d,%d,%d",
                   tempnxtptr->x,tempnxtptr->y,tempnxtptr->z);
       delete_agenda_node(Map[tempnxtptr->x][tempnxtptr->y][tempnxtptr-
>z].ptrtoagenda);
        Map[tempnxtptr->x][tempnxtptr->y][tempnxtptr->z].ptrtoagenda = tempnxtptr;
        Map[tempnxtptr->x][tempnxtptr->y][tempnxtptr->z].condition = FRONTIER;
        Map[tempnxtptr->x][tempnxtptr->y][tempnxtptr->z].cumvertcost =
                                                    tempnxtptr->cumcost;
      }
      else
      {
        free(tempnxtptr);
        if (verbose == TRUE)
          printf("\n  Not Replacing frontier node %d,%d,%d",
            tempnxtptr->x,tempnxtptr->y,tempnxtptr->z);
        return;
      }
    }
    /* if node is not even on the agenda then change its state and add */
    /* it to agenda list */
    if (Map[tempnxtptr->x][tempnxtptr->y][tempnxtptr->z].condition == NOTEVAL)
    {
      Map[tempnxtptr->x][tempnxtptr->y][tempnxtptr->z].condition = FRONTIER;
      Map[tempnxtptr->x][tempnxtptr->y][tempnxtptr->z].cumvertcost =
        tempnxtptr->cumcost;
      if (verbose == TRUE)
        printf("\n  Changing Node %d,%d,%d to FRONTIER",
           tempnxtptr->x,tempnxtptr->y,tempnxtptr->z);
    }
    /* add to list ordered by criteria value */
    if (ash != NULL)
    {
      if (verbose == TRUE)
        printf("\n  Adding node %d,%d,%d to agenda",
           tempnxtptr->x,tempnxtptr->y,tempnxtptr->z);
      /* cycle through nodes in list to place in order by criteria */
      while (curragendaptr != NULL)
      {
        /* lowest criteria value in list */
        if (tempnxtptr->criteria < ash->criteria)
        {
          /* annotate previous node array */
          mark_vertex_path(tempcurptr,tempnxtptr);
          if(verbose == TRUE)
            printf("\n  Inserting vertex %d,%d,%d at head of agenda list",
                      tempnxtptr->x,tempnxtptr->y,tempnxtptr->z);
          tempnxtptr->nextasnodeptr = ash;
          ash = tempnxtptr;
          /* clean up pointers */
          free(tempcurptr);
          tempnxtptr = NULL;
          tempcurptr = NULL;
          return;
        }
        /* advance pointer to second node on agenda */
        curragendaptr = curragendaptr->nextasnodeptr;
        /* highest criteria value in list */
        /* not lowest criteria value in list */
        if (curragendaptr == NULL ||
            (tempnxtptr->criteria <= curragendaptr->criteria &&
             tempnxtptr->criteria >= prevagendaptr->criteria))
        {
          /*annotate previous node array */
          mark_vertex_path(tempcurptr,tempnxtptr);
          if(verbose == TRUE)
            printf("\n  Inserting vertex %d,%d,%d within agenda list",
                      tempnxtptr->x,tempnxtptr->y,tempnxtptr->z);
          tempnxtptr->nextasnodeptr = curragendaptr;
          prevagendaptr->nextasnodeptr = tempnxtptr;
          /* clean up pointers */
          free(tempcurptr);
          tempnxtptr = NULL;
          tempcurptr = NULL;
          return;
        }
        prevagendaptr = prevagendaptr->nextasnodeptr;
      }
    }
    /* agenda list is empty */
    if (ash == NULL)
    {
      /*annotate previous node array */
      mark_vertex_path(tempcurptr,tempnxtptr);
      if(verbose == TRUE)
        printf("\n  Agenda list is empty. Inserting %d,%d,%d ",
                  tempnxtptr->x,tempnxtptr->y,tempnxtptr->z);
      ash = tempnxtptr; /* assign head ptr to first node in list */
      /* clean up pointers */
          free(tempcurptr);
          tempnxtptr = NULL;
          tempcurptr = NULL;
    }

  }

  /**************************************************************************/
  /*DELETE_AGENDA_NODE..Locates node in agenda list and deletes             */
  /**************************************************************************/
  void delete_agenda_node(agendapointer)

    ASNODELINK agendapointer;

  {
```

```
  ASNODELINK curptr,prevptr;

  curptr = ash;     /* point to top of list */
  prevptr = curptr;

  /* if first node on agenda list */
  if (ash == agendapointer)
  {
    ash = ash->nextasnodeptr;
    free(curptr);
    return;
  }
  /* not first node on agenda list */
  while (curptr != NULL || curptr != agendapointer)
  {
    curptr = curptr->nextasnodeptr;
    if (curptr == agendapointer)
    {
      prevptr->nextasnodeptr = curptr->nextasnodeptr;
      free(curptr);
      return;
    }
    prevptr = curptr;
  }
  return;

}

/***************************************************************************/
/*GET_PRIORITY_NODE_AND_DELETE_FROM_AGENDA..Points to the principle node */
/*in agenda, returns it, and deletes it from the agenda                   */
/***************************************************************************/
ASNODELINK get_principle_node_and_delete_from_agenda()

{

  ASNODELINK principlenodeptr;

  if(ash == NULL)
    printf("\n  Agenda is empty, cannot get principle node");
  principlenodeptr = ash;
  ash = principlenodeptr->nextasnodeptr;
  principlenodeptr->nextasnodeptr = NULL;
  Map[principlenodeptr->x][principlenodeptr->y][principlenodeptr->z].condition =
                                                               CHECKED;
  if(verbose == TRUE)
    printf("\n  Getting principle node %d,%d,%d from agenda list",
           principlenodeptr->x,principlenodeptr->y,principlenodeptr->z);
  return principlenodeptr;

}

/***************************************************************************/
/*DELETE_AGENDA_LIST..Deletes entire agenda list to clean up              */
/***************************************************************************/
void delete_agenda_list()

{

  ASNODELINK tempptr;

  if(verbose == TRUE)
    printf("\n  Goal reached, deleting remaining agenda list");
  while(ash != NULL)
  {
    tempptr = ash;
    ash = ash->nextasnodeptr;
    tempptr->nextasnodeptr = NULL;
    if(verbose == TRUE) printf("\n  Deleting vertex %d,%d,%d from agenda list",
                                tempptr->x,tempptr->y,tempptr->z);
    free(tempptr);
  }

}

/***********************************************************************/
/*MARK_VERTEX_PATH..Annotates previous node in as array node          */
/***********************************************************************/
void mark_vertex_path(prevasnode,currentasnode)

  ASNODELINK prevasnode,currentasnode;

{

  if(verbose == TRUE)
    printf("\n  Vertex array position %d,%d,%d.prev gets %d,%d,%d",
              currentasnode->x,currentasnode->y,currentasnode->z,
              prevasnode->x,prevasnode->y,prevasnode->z);
  Map[currentasnode->x][currentasnode->y][currentasnode->z].xprev =
                                                   prevasnode->x;
  Map[currentasnode->x][currentasnode->y][currentasnode->z].yprev =
                                                   prevasnode->y;
  Map[currentasnode->x][currentasnode->y][currentasnode->z].zprev =
                                                   prevasnode->z;
  /* points to current node on the agenda */
  Map[currentasnode->x][currentasnode->y][currentasnode->z].ptrtoagenda =
                                               currentasnode;

}

/***********************************************************************/
/*CREATE_PATH_LIST..Creates and builds a path to goal list             */
/*head of list is first node after vehicle, last node is goal          */
/***********************************************************************/
NEWPATHLINK create_path_list(goalptr)

  ASNODELINK goalptr;

{

  NEWPATHLINK pthd,newptr;

  if(verbose == TRUE)
    printf("\n  Goal reached, creating path list as follows:");
  /* put goal position in list */
  newptr = (Newpath*)malloc(sizeof(Newpath));
  newptr->xpath = goalptr->x;
  newptr->ypath = goalptr->y;
  newptr->zpath = goalptr->z;
  newptr->nextnewnode = NULL;
  if (storemissiondata == TRUE)
    fprintf(missionofp,"%c %d %d %d %d %d\n",'$',
                              newptr->xpath,
                              newptr->ypath,
                              newptr->zpath,
                              0,
                              6);
  pthd = newptr;
  if(verbose == TRUE)
    printf("\n  %d,%d,%d",newptr->xpath,newptr->ypath,newptr->zpath);
  /* trace back through path placing in list until AUV reached */
  /* printf("\n\n\n\nAUV POSITION IS %d,%d,%d\n\n\n\n",
                   Auv.grid.xpos,Auv.grid.ypos,Auv.grid.zpos);
   printf("\n\n\n\nARRAY PREV POSITION IS %d,%d,%d\n\n\n\n",
          Map[newptr->xpath][newptr->ypath][newptr->zpath].xprev,
            Map[newptr->xpath][newptr->ypath][newptr->zpath].yprev,
            Map[newptr->xpath][newptr->ypath][newptr->zpath].zprev); */
  while (((Map[newptr->xpath][newptr->ypath][newptr->zpath].xprev !=
             Auv.grid.xpos) ||
            (Map[newptr->xpath][newptr->ypath][newptr->zpath].yprev !=
             Auv.grid.ypos) ||
            (Map[newptr->xpath][newptr->ypath][newptr->zpath].zprev !=
             Auv.grid.zpos)))
  {
    newptr = (Newpath*)malloc(sizeof(Newpath));
    newptr->xpath = Map[pthd->xpath][pthd->ypath][pthd->zpath].xprev;
    newptr->ypath = Map[pthd->xpath][pthd->ypath][pthd->zpath].yprev;
    newptr->zpath = Map[pthd->xpath][pthd->ypath][pthd->zpath].zprev;
    newptr->nextnewnode = pthd;
    if (storemissiondata == TRUE)
      fprintf(missionofp,"%c %d %d %d %d %d\n",'$',
                              newptr->xpath,
                              newptr->ypath,
                              newptr->zpath,
                              0,
                              6);
    pthd = newptr;
    if(verbose == TRUE)
      printf("\nPath  %d,%d,%d",newptr->xpath,newptr->ypath,newptr->zpath);
  }
  return pthd;

}

/****************************************************************************/
/*ODD..Determines if a value is odd                                        */
/****************************************************************************/
int odd(value)

  int value;
{

  if (value % 2 == 0)
    return EVEN;
  else
    return ODD;

}

/****************************************************************************/
/*EVEN..Determines if a value is even                                      */
/****************************************************************************/
int even(value)

  int value;

{

  if (value % 2 == 0)
    return ODD;
  else
    return EVEN;

}

/****************************************************************************/
/*COMPUTE_DIST..Determines the distance between two nodes                  */
/****************************************************************************/
float compute_dist(xloc,xpos,yloc,ypos,zloc,zpos)

  int xloc,xpos,yloc,ypos,zloc,zpos;

{
  double dist;

  dist = sqrt((double)(((xloc - xpos)*(xloc - xpos)) +
                       ((yloc - ypos)*(yloc - ypos)) +
                       ((zloc - zpos)*(zloc - zpos))));
  if(verbose == TRUE)
    printf("\nThe distance between %d,%d,%d and node %d,%d,%d is %f",
                  xloc,yloc,zloc,xpos,ypos,zpos,(float)dist);
  return (float)dist;

}

/**************************************************************************/
/*INSIDE_AREA..Determines if a node is inside the search area            */
/**************************************************************************/
int inside_area(x,y,z)

  int x,y,z;

{

  if (x >= 0 && x < MAXX && y >= 0 && y < MAXY && z >= 0 && z < MAXZ)
    return TRUE;
  else
    return FALSE;

}

/**************************************************************************/
/*IS_ADJACENT..Determine if a node is adjacent to the vehicle            */
/**************************************************************************/
int is_adjacent(a,b,c,d)

  int a,b,c,  /* node location being analyzed */
      d;      /* state of location being anazyzed */

{

  int e,f,g;  /* holder for vehicle position */

  e = Auv.grid.xpos;
  f = Auv.grid.ypos;
  g = Auv.grid.zpos;
  if((a <= e+1 && a >= e-1) &&
     (b <= f+1 && b >= f-1) &&
     (c <= g+1 && c >= g-1) &&
     (c != AUV))
  {
    return TRUE;
  }
  else
  {
    return FALSE;
  }

}

/**************************************************************************/
/*ADJACENT_NODE..Returns true of one node is adjacent to the other       */
/**************************************************************************/
int adjacent_node(x,y,z,nextx,nexty,nextz)

  int x,y,z,nextx,nexty,nextz;

{

  if((nextx <= x+1 && nextx >= x-1) &&
     (nexty <= y+1 && nexty >= y-1) &&
     (nextz <= z+1 && nextz >= z-1))
    return TRUE;
  else
    return FALSE;

}

/*************************************************************************/
/*CLEAN_UP..Clean up after running program                              */
/*************************************************************************/
void clean_up()

{

  print_active_list();
  if (storemissiondata == TRUE)
    fclose(missionofp);                /* close the mission file */
    fclose(trackofp);                  /* close the track file */

}

/*************************************************************************/
/*TEST_ADVANCE..Test routine for advancing AUV through world. Does not   */
/*currently employ a-star or other search.                               */
/*************************************************************************/
void test_advance(testnode)

  NODELINK testnode;

{

  /* clean up old AUV position */
  Wld[Auv.grid.xpos][Auv.grid.ypos][Auv.grid.zpos].state = VISITED;
  Wld[Auv.grid.xpos][Auv.grid.ypos][Auv.grid.zpos].visited = TRUE;
  /* send updated node data from former AUV position to file */
  /* $ represents vehicle data */
  if (storemissiondata == TRUE)
  {
  fprintf(missionofp,"%c %d %d %d %d %d\n",'$',
          Wld[Auv.grid.xpos][Auv.grid.ypos][Auv.grid.zpos].grid.xpos,
          Wld[Auv.grid.xpos][Auv.grid.ypos][Auv.grid.zpos].grid.ypos,
          Wld[Auv.grid.xpos][Auv.grid.ypos][Auv.grid.zpos].grid.zpos,
          Wld[Auv.grid.xpos][Auv.grid.ypos][Auv.grid.zpos].dir,
          Wld[Auv.grid.xpos][Auv.grid.ypos][Auv.grid.zpos].state);
  }
  /* update AUV position and status */
  Auv.grid.xpos = testnode->x;
  Auv.grid.ypos = testnode->y;
  Auv.grid.zpos = testnode->z;
  Auv.x = testnode->x;
  Auv.y = testnode->y;
  Auv.z = testnode->z;
  Wld[testnode->x][testnode->y][testnode->z].state = AUV;
  if(verbose == TRUE)
    printf("\nAUV position = %d,%d,%d",
                    Wld[testnode->x][testnode->y][testnode->z].grid.xpos,
                    Wld[testnode->x][testnode->y][testnode->z].grid.ypos,
                    Wld[testnode->x][testnode->y][testnode->z].grid.zpos);
  /* send updated AUV information to file */
  if(storemissiondata == TRUE)
  /* $ represents vehicle data */
  fprintf(missionofp,"%c %d %d %d %d %d\n",'$',
          Wld[Auv.grid.xpos][Auv.grid.ypos][Auv.grid.zpos].grid.xpos,
          Wld[Auv.grid.xpos][Auv.grid.ypos][Auv.grid.zpos].grid.ypos,
          Wld[Auv.grid.xpos][Auv.grid.ypos][Auv.grid.zpos].grid.zpos,
          Wld[Auv.grid.xpos][Auv.grid.ypos][Auv.grid.zpos].dir,
          Wld[Auv.grid.xpos][Auv.grid.ypos][Auv.grid.zpos].state);
  /* remove visited node from visit list */
  delete_from_visit_list(testnode->x,testnode->y,testnode->z);

}

/***************************************************************************/
/*PRINT_ACTIVE_LIST..Prints contents of active node list to screen        */
/***************************************************************************/
void print_active_list()

{

  SUBAREALINK ptrsubarea;
  NODELINK ptrnode;

  ptrsubarea = zh;
  while(ptrsubarea != NULL)
  {
    if(verbose == TRUE) printf("Subarea is %d\n",ptrsubarea->subareanum);
    ptrnode = ptrsubarea->nodelistptr;
    while(ptrnode != NULL)
    {
      if(verbose == TRUE) printf(" Node is %d,%d,%d\n",
                               ptrnode->x,ptrnode->y,ptrnode->z);
      ptrnode = ptrnode->nextnodeptr;
    }
    ptrsubarea = ptrsubarea->nextsubareaptr;
  }

}

/***************************************************************************/
/*PRINT_AGENDA_LIST..Prints contents of active node list to screen        */
/***************************************************************************/
void print_agenda_list()

{

  ASNODELINK ptrasnode;

  ptrasnode = ash;
  while(ptrasnode != NULL)
  {
     if(verbose == TRUE)  printf("\n    Agenda node is %d,%d,%d",
                            ptrasnode->x,ptrasnode->y,ptrasnode->z);
    ptrasnode = ptrasnode->nextasnodeptr;
  }

}

/***************************************************************************/
/*PRINT_PATH_LIST..Prints contents of active node list to screen        */
/***************************************************************************/
void print_path_list(pathptr)

  NEWPATHLINK pathptr;
{

  while(pathptr != NULL)
  {
    if(verbose == TRUE)   printf("\n    Path node is %d,%d,%d",
                            pathptr->xpath,pathptr->ypath,pathptr->zpath);
    pathptr = pathptr->nextnewnode;
  }

}
```

APPENDIX C: TWO-DIMENSIONAL GRAPH SEARCH EVALUATION TOOL SOURCE CODE

```
/****************************************************************************
Title:  search2d.h
Author: Mark Compton
Course: Thesis
Date:   12 Mar 92

Description: This program builds a two dimensional search area designed to
             test graph type searches for theses work. The tool is called
             2-D GRAPH SEARCH EVALUATION TOOL.

Support: search.c (contains main menu for building tool)
         search_support.c (contains support routines for building tool)
****************************************************************************/

/*Preprocessing Directives***************************************************/
#define BOXWIDTH  15.0  /* box width               */
#define BOXHEIGHT 15.0  /* box height              */
#define MAXX 63         /* number of nodes in x direction */
#define MAXY 63         /* number of nodes in y direction */
#define SUBX 7          /* number of subareas in x direction */
#define SUBY 7          /* number of subareas in y direction */
#define SUBXNODES 9     /* number of nodes in each subarea in x direction */
#define SUBYNODES 9     /* number of nodes in each subarea in y direction */
#define MAXSTRING 20    /* for file names */

#define ODD  1          /* defines if a value is odd */
#define EVEN 0          /* defines if a value is even */

#define SLOW 1000000          /* defines playback speeds */
#define MEDIUM 150000
#define FAST 15000

/* defines states of a node */
#define FREE      0     /* nothing at node */
#define OBSTACLE 1      /* obstacle at node */
#define AUV      2      /* vehicle (Autonomous Underwater Vehicle) */
#define ADJACENT 3      /* node adjacent to vehicle */
#define ACTIVE   4      /* detected, non-object, not visited */
#define VISITED  5      /* node has been previously visited */
#define ASPATH   6      /* local path selected by A-Star Search */

/* define colors of objects in search world */
#define FREECOLOR       RGBcolor(155,155,200);
#define OBSTACLECOLOR   RGBcolor(0,0,0);
#define AUVCOLOR        RGBcolor(0,0,255);
#define ADJACENTCOLOR   RGBcolor(110,45,0);
#define ACTIVECOLOR     RGBcolor(255,255,0);
#define VISITEDCOLOR    RGBcolor(255,255,255);
#define DIRECTIONCOLOR  RGBcolor(255,255,0);
#define ASCOLOR         RGBcolor(255,0,0);

#define STOREOBSTACLES 10       /* menu pick for storing obstacle data */
#define RETRIEVEOBSTACLES 20    /* menu pick for retrieving obstacle data */
#define RETRIEVEMISSION 30      /* menu pick for vehicle track */
#define PLAYBACKMISSION 40      /* menu pick for mission playback */
#define PAUSEMISSION 42         /* menu pick to pause mission playback */
#define RESUMEMISSION 44        /* menu pick to resume mission playback */
#define CLEARAREA 50            /* menu pick to exit program */
```

```
#define PLAYBACKSPEEDSLOW 60     /* menu pick for playback speeds */
#define PLAYBACKSPEEDMEDIUM 61   /* menu pick for playback speeds */
#define PLAYBACKSPEEDFAST 62     /* menu pick for playback speeds */
#define EXIT          70         /* menu pick to exit program */

/* defines direction of vehicle */
#define NODIR 0
#define N  1
#define NE 2
#define E  3
#define SE 4
#define S  5
#define SW 6
#define W  7
#define NW 8

*Globals******************************************************************/
char outnode[MAXSTRING];        /* output file for nodes */
FILE *obstacleofp;              /* pointer for sending obstacles to file */
FILE *obstacleifp;              /* pointer for receiving obstacles from file */
FILE *missionifp;               /* pointer for receiving mission from file */
char inobstacle[MAXSTRING];     /* input file name for obstacles */
char inmission[MAXSTRING];      /* input file name for mission */
int mouse_select = FALSE;       /* toggle for mouse selections */
int left_mouse_select = FALSE;  /* toggle for mouse selections */
int playthemission = FALSE;     /* flag for playback of mission */
int playbackspeed = MEDIUM;     /* initializes playback speed */
/*Structures***************************************************************/
struct location
{
  int xpos,ypos;
        /* grid coordinates */
};

typedef struct location Location;

struct node                 /* basic structure of a search area node */
{
  Location grid;            /* grid coordinates of node */
  int area;                 /* subarea node is located in */
  double x,y;               /* world position of node */
  int dir;                  /* direction node is looking */
  int state;                /* status of node FREE, OBSTACLE, AUV*/
};
  typedef struct node Node;

Node Wpt[MAXX][MAXY];       /* 2-D array for storing search area nodes */

struct missiondata
{
  char info;                              /* type of information for display */
  Location grid;                          /* grid coordinates of node */
  int dir;                                /* direction vehicle is looking */
  int state;                              /* status of node */
  float distrav;
  int auvsteps;
  float eratio;
  float estepratio;
  struct missiondata *nextmissiondata; /* points to next mission data */
};
  typedef struct missiondata Missiondata;
  typedef Missiondata *MISSIONLINK;       /* pointer to Missiondata */
  MISSIONLINK head;                       /* points to head of list */
  MISSIONLINK current;                    /* points to item in mission list */
  MISSIONLINK prev;                       /* points to previous in mission list */
```

```
/***************************************************************************
Title:  search2d.c
Author: Mark Compton
Course: Thesis
Date:   12 Mar 92

Description: This program builds a two dimensional search area designed to
             test graph type searches for theses work.
***************************************************************************/

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "gl.h"
#include "device.h"
#include "search2d.h"
#include "search_support2d.c"

main ()
{

  char  response = 'n';
  char  answer   = 'n';
  char  acknowledgement = 'n';
  int mainmenu;
  short value;   /* the value from the Event Queue */
  float wx,wy;   /* world coord location of the mouse */
  long  i,j;     /* counters for send to file */
  int hititem;   /* variable holding hit name */

    /* Setup for retrieving mission data */
    printf("\n\nMission data may be retrieved from a file ");
    printf("\nthen displayed by selecting Retrieve Mission in");
    printf("\nthe menu.\n\n ");
    printf("Will you be retrieving mission data from a file?  ");
    scanf("%c",&response);
    if (response == 'y' || response == 'Y')
    {
      printf("\n\nPlease enter the mission input file name: \n");
      scanf("%s",inmission);
    }
    /* Setup for retrieving obstacle data */
    scanf("%c",&answer); /* hack to clear carrage return from buffer */
    printf("\nObstacle data may be retrieved from a file ");
    printf("\nthen displayed by selecting Retrieve Obstacle in");
    printf("\nthe menu.\n\n ");
    printf("Will you be retrieving obstacle data from a file?  ");
    scanf("%c",&answer);
    if (answer == 'y' || answer == 'Y')
    {
      printf("\n\nPlease enter the obstacle input file name: \n");
      scanf("%s",inobstacle);
    }
   /* initialize the graphics system */
   initialize();
   /* make the popup menus */
   mainmenu = makethemenus();
   /* set the world coordinate system */
   ortho2(0.0,(float)XMAXSCREEN,0.0,(float)YMAXSCREEN);
   /* stay inside this display loop until the
      Window Manager terminates us...
   */
   /* first time thru for drawing area */
   drawthearea();
   while(TRUE)
   {
      /* is there anything on the Event Queue? */
      while(qtest() != 0)
```

```
         {
            switch(qread(&value))
            {
               case REDRAW:
                           reshapeviewport();
                           break;
               case LEFTMOUSE:
                           /* if the button has been depressed */
                           if(value == 1)
                           {
                             /* compute the world coordinate of the mouse */
                              wx = getvaluator(MOUSEX);
                              wy = getvaluator(MOUSEY);
                              /* left_mouse_select = TRUE; */
                              check_node_info(wx,wy);
                           }
                           break;
               case MIDDLEMOUSE:
                           /* if the button has been depressed */
                           if(value == 1)
                           {
                              /* compute the world coordinate of the mouse */
                              wx = getvaluator(MOUSEX);
                              wy = getvaluator(MOUSEY);
                              /* mouse_select = TRUE; */
                              put_obstacle(wx,wy);
                           }
                           break;
               case MENUBUTTON:

                  if(value == 1)
                  {
                     hititem = dopup(mainmenu);

                     processmenuhit(hititem);
                     /* value = 0; */
                  }
                  break;
               default:
                           break;
            } /* end switch */
         } /* endif there was something on the queue */
         if (playthemission == TRUE)
         {
            playbackmission();
         }
      } /* end while loop */

   }

/**************************************************************************
Title:  search_support.c
Author: Mark Compton
Course: Thesis
Date:   12 Mar 92

Description: This program contains routines which support the building of
             a two dimensional graph search evaluation tool.
**************************************************************************/

#include <stdio.h>
#include <stdlib.h>
#include <math.h>

/*************************************************************************/
/*INITIALIZE..Establish IRIS graphics preferences                         */
/*************************************************************************/
initialize()
```

```
{
   /* set up a preferred size for the window */
   prefsize(XMAXSCREEN+1,YMAXSCREEN+1);
   /* set up a preferred position for the window */
   prefposition(0,XMAXSCREEN,0,YMAXSCREEN);
   /* open a window for the program */
   winopen("pick");
   /* put a title on the window */
   wintitle("Pick -- Middle Mouse is the Pick ");
   /* set color type */
   RGBmode();
   /* put the IRIS into double buffer mode */
   /* doublebuffer(); */
   singlebuffer();
   /* configure the IRIS (means use the above command settings) */
   gconfig();
   /* queue the redraw device */
   qdevice(REDRAW);
   /* queue the menubutton */
   qdevice(MENUBUTTON);
   /* queue the left mouse button */
   qdevice(LEFTMOUSE);
   /* queue the middle mouse button */
   qdevice(MIDDLEMOUSE);
   /* select flat shading */
   /* only works on the 4D*/
#ifdef FLAT
   shademodel(FLAT);
#endif FLAT

}

/***************************************************************************/
/*MAKETHEMENUS..Performs all the menu construction call                    */
/***************************************************************************/
int makethememenus()

{

   int topmenu;   /* top level menu's name */

   /* build the top level menu */
   topmenu = defpup("Search Menu %t | Store Obstacles %x10 \
                                   | Retrieve Obstacles %x20 \
                                   | Retrieve Mission Data %x30 \
                                   | Playback Mission %x40 \
                                   | Pause Mission Playback %x42 \
                                   | Resume Mission Playback %x44 \
                                   | Clear Area %x50 \
                                   | Playback Slow %x60 \
                                   | Playback Medium %x61 \
                                   | Playback Fast %x62 \
                                   | Exit %x70 ");
   /* return the name of this menu */
   return(topmenu);

}

/***************************************************************************/
/*PROCESSMENUHIT..Performs all the menu actions                            */
/***************************************************************************/
processmenuhit(hititem)

int hititem;   /* item hit on the popup menus */

{

   switch(hititem)
   {
         case STOREOBSTACLES:
                      storedata();
                      break;
         case RETRIEVEOBSTACLES:
                      retrieveobstacledata();
                      break;
         case RETRIEVEMISSION:
                      retrievemission();
                      break;
         case PLAYBACKMISSION:
                      playthemission = TRUE;
                      current = head;
                      prev = head;
                      break;
         case PAUSEMISSION:
                      playthemission = FALSE;
                      break;
         case RESUMEMISSION:
                      playthemission = TRUE;
                      break;
         case CLEARAREA:
                      drawthearea();
                      playthemission = FALSE;
                      break;
         case PLAYBACKSPEEDSLOW:
                      playbackspeed = SLOW;
                      break;
         case PLAYBACKSPEEDMEDIUM:
                      playbackspeed = MEDIUM;
                      break;
         case PLAYBACKSPEEDFAST:
                      playbackspeed = FAST;
                      break;
         case EXIT:
                      exit(0);
                      break;
         default:
                      break;
      } /* end switch */

}

/****************************************************************************/
/*DRAWTHEAREA..Draws the graph search environment                          */
/****************************************************************************/
drawthearea()

{

   float startx,starty;          /* starting x and y of the grid */
   float x,y,c,d;                /* temp coord loc */
   long i,j,a,b;                 /* temp loop counters */

   /* draw the background color */
   RGBcolor(0,255,255);
   clear();
   /* put up some help text */
   RGBcolor(255,0,0);
   cmov2i(10,1000);
   charstr("GRAPH SEARCH EVALUATION TOOL");
   RGBcolor(255,0,0);
   cmov2i(10,998);
   charstr("_____ ______ __________ ____");
   RGBcolor(0,0,0);
   cmov2i(10,985);
   charstr("             by Mark Compton");
   RGBcolor(0,0,255);
```

```
cmov2i(1000,1000);
charstr("Mouse Functions");
cmov2i(1000,998);
charstr("_____ _________");
cmov2i(1000,985);
charstr("Left:   Node Information");
cmov2i(1000,970);
charstr("Middle: Build Obstacle");
cmov2i(1000,955);
charstr("Right:  Menu");
cmov2i(1000,935);
charstr("Mouse Select Data");
cmov2i(1000,933);
charstr("_____ ______ ____");
cmov2(1000,920);
charstr("x =      y =   ");
cmov2(1000,905);
charstr("state =   ");
cmov2(1000,890);
charstr("area =   ");
RGBcolor(0,0,255);
cmov2i(1000,100);
charstr("Statistics");
cmov2(1000,98);
charstr("__________");
cmov2(1000,80);
charstr("Graph steps = ");
cmov2(1000,65);
charstr("Distance traveled = ");
cmov2(1000,50);
charstr("Effective dist ratio = ");
/* compute the start of the grid */
startx = ((XMAXSCREEN - MAXX * BOXWIDTH)/2.0)-145.0;
starty = ((YMAXSCREEN - MAXY * BOXHEIGHT)/2.0)-20.0;
/* draw background subareas */
  for(a=0; a < MAXY/SUBYNODES; ++a)
  {
    for(b=0; b < MAXX/SUBXNODES; ++b)
    {
       /* compute the color */
      if (odd(a) == 0)
      {
        if (odd(b) == 0)
        {
        RGBcolor(0,255,0);
        }
        if (odd(b) == 1)
        {
        RGBcolor(155,0,155);
        }
       }
      if (odd(a) == 1)
      {
        if (odd(b) == 0)
        {
        RGBcolor(155,0,155);
        }
        if (odd(b) == 1)
        {
        RGBcolor(0,255,0);
        }
       }
       /* compute the lowerleft of the box to fill */
       c = (startx + a*BOXWIDTH*SUBXNODES);
       d = (starty + b*BOXHEIGHT*SUBYNODES);
       /* draw the filled rectangles */
       rectf(c,d,c+BOXWIDTH*SUBXNODES
               ,d+BOXHEIGHT*SUBYNODES);
```

```
          }
        }
      /* draw boarder */
      RGBcolor(0,0,0);
      linewidth(10);
      rect(startx-2.5,starty-2.5,
           startx+5.0+(BOXWIDTH*MAXX),starty+5.0+(BOXHEIGHT*MAXY));
      /* draw all the nodes */
      for(j=0; j < MAXY; j=j+1)
      {
         for(i=0; i < MAXX; i=i+1)
         {
            /* compute the lowerleft of the box to fill */
            x = (startx + i*BOXWIDTH) + 3.0;
            y = (starty + j*BOXHEIGHT) + 3.0;
            /* fill in Node structure */
            Wpt[i][j].x = x+BOXWIDTH/2.0;
            Wpt[i][j].y = y+BOXWIDTH/2.0;
            Wpt[i][j].state = FREE;
            Wpt[i][j].grid.xpos = i;
            Wpt[i][j].grid.ypos = j;
            Wpt[i][j].area = subarea(i,j);
            /* determine color of node */
            if (Wpt[i][j].state == OBSTACLE)
              OBSTACLECOLOR;
            if (Wpt[i][j].state == FREE)
              FREECOLOR;
            if(Wpt[i][j].state == AUV)
              AUVCOLOR;
            if(Wpt[i][j].state == ADJACENT)
              ADJACENTCOLOR;
            if(Wpt[i][j].state == ASPATH)
              ASCOLOR;
            /* draw the filled rectangles */
            rectf(x,y,x+BOXWIDTH-3.0,y+BOXHEIGHT-3.0);
         }
      }

}

/*************************************************************************/
/*ODD..Determines if a value is odd or even                              */
/*************************************************************************/
odd(value)

  int value;

{

  if (value % 2 == 0)
    return EVEN;
  else
    return ODD;

}

/*************************************************************************/
/*INSIDE..determines if (x,y) is inside the box defined by the           */
/*coordinates (xmin,ymin)-(xmax,ymax)                                    */
/*************************************************************************/
int inside(x,y,xmin,ymin,xmax,ymax)

float x,y;   /* location of the cursor */
float xmin,ymin,xmax,ymax;   /* bounding box to check if cursor is inside */

{
   if((xmin <= x) && (x <= xmax) && (ymin <= y) && (y <= ymax))
   {
      return(TRUE);
   }
   else
   {
      return(FALSE);
   }

}

/****************************************************************************/
/*STOREDATA..Stores node data to file if requested                          */
/****************************************************************************/
storedata()

{

  int i,j;

  obstacleofp = fopen("obstacle_file","w");
  for(i=0; i < MAXY; i=i+1)
  {
    for(j=0; j < MAXX; j=j+1)
    {
      if(Wpt[i][j].state == OBSTACLE)
      {
      fprintf(obstacleofp,"%d %d\n",Wpt[i][j].grid.xpos,
                                     Wpt[i][j].grid.ypos);
       }
    }
  }
  fclose(obstacleofp);

}

/****************************************************************************/
/*RETRIEVEOBSTACLEDATA..Retrieves obstacle data from file if requested      */
/****************************************************************************/
retrieveobstacledata()

{

  int x,y;

  obstacleifp = fopen(inobstacle,"r");
  while(fscanf(obstacleifp, "%d%d", &x,&y) != EOF)
  {
    Wpt[x][y].state = OBSTACLE;
    draw_node(Wpt[x][y].grid.xpos,Wpt[x][y].grid.ypos);
  }
  fclose(obstacleifp);

}

/****************************************************************************/
/*RETRIEVEMISSION..Retrieves mission data from file if requested            */
/*Stores mission data in linked list for future use                         */
/****************************************************************************/
retrievemission()

{

  int x,y,nodedir,nodestate,au;
  float di,er,es;
  char w; /* w indicates type of line to follow */
  MISSIONLINK currentptr,previousptr;

  /* draw the print information background */
  RGBcolor(0,0,0);
  rectf(270,420,730,580);
  RGBcolor(255,0,0);
  rectf(300,450,700,550);
  RGBcolor(0,0,0);
  cmov2i(400,500);
  charstr("STANDBY----LOADING MISSION");
  missionifp = fopen(inmission,"r");
  fscanf(missionifp, "%c%d%d%d%d%f%d%f%f",&w,&x,&y,&nodedir,&nodestate,
                    &di,&au,&er,&es);
      head = malloc(sizeof(Missiondata));
      head -> info       = w;
      head -> grid.xpos  = x;
      head -> grid.ypos  = y;
      head -> dir        = nodedir;
      head -> state      = nodestate;
      head -> distrav    = di;
      head -> auvsteps   = au;
      head -> eratio     = er;
      head -> estepratio = es;
      previousptr = head;
  while (fscanf(missionifp, "%c%d%d%d%d%f%d%f%f",
              &w,&x,&y,&nodedir,&nodestate,
              &di,&au,&er,&es) != EOF)
  {
      currentptr = malloc(sizeof(Missiondata));
      currentptr -> info      = w;
      currentptr -> grid.xpos = x;
      currentptr -> grid.ypos = y;
      currentptr -> dir       = nodedir;
      currentptr -> state     = nodestate;
      currentptr -> distrav    = di;
      currentptr -> auvsteps   = au;
      currentptr -> eratio     = er;
      currentptr -> estepratio = es;
      previousptr -> nextmissiondata = currentptr;
      previousptr = currentptr;
  }

  fclose(missionifp);
  drawthearea(); /* get rid of message */

}

/***************************************************************************/
/*KILL_TIME..Kills processor time for visual purposes in drawing          */
/***************************************************************************/
kill_time()

{

int a,b,c;

  for(a=0;a<playbackspeed;++a)
    b = 10000000;
    c = a*b;

}

/***************************************************************************/
/*PLAYBACKMISSION..Playback of mission data                               */
/***************************************************************************/
playbackmission()

{

  if(current == NULL)
    playthemission = FALSE;
  else
  {
    if (current != NULL)
    {
      /* world information to follow */
      if(current->info == '%' &&
         Wpt[current->grid.xpos][current->grid.ypos].state != current->state)
      {
        /* assign input state to node */
        Wpt[current->grid.xpos][current->grid.ypos].state = current->state;
        draw_node(Wpt[current->grid.xpos][current->grid.ypos].grid.xpos,
                  Wpt[current->grid.xpos][current->grid.ypos].grid.ypos);
      }
      /* search path information to follow */
      if(current->info == '$' ||
         current->info == '+')
      {
          /* assign input direction to node */
          Wpt[current->grid.xpos][current->grid.ypos].dir = current->dir;
          /* assign input state to node */
          Wpt[current->grid.xpos][current->grid.ypos].state = current->state;
          /* update vehicle graph step count */
          if (current->state == 2)
          {
            update_node_info(current->grid.xpos,current->grid.ypos,
                             current->distrav,current->auvsteps,
                             current->eratio);
          }
          draw_node(Wpt[current->grid.xpos][current->grid.ypos].grid.xpos,
                    Wpt[current->grid.xpos][current->grid.ypos].grid.ypos);
          kill_time(); /* used for better visual presentation */
      }
      prev = current;
      current = current->nextmissiondata;
      /* free(prev); use only if desire to delete list */
    }
  }

}

/*************************************************************************/
/*SUBAREA..Determines which subarea a node is in                          */
/*************************************************************************/
subarea(a,b)

int a,b; /* feed in the grid coordinates */

{

  int i,j,row,column,area;

  for (i = 1; i <= SUBY; ++i)              /* step thru subarea rows */
  {
    for (j = 1; j <= SUBX; ++j)            /* step thru subarea columns */
    {
      if (b >= (i-1)*SUBYNODES && b < i*SUBYNODES)
      {
        row = i;
      }
      if (a >= (j-1)*SUBXNODES && a < j*SUBXNODES)
      {
        column = j;
      }
      if (odd(row))
      {
        area = ((row-1)*SUBY)+column;
      }
      else
      {
          area = ((row*SUBY)+1)-column;
        }
      }
    }
    return area;

}

/***************************************************************************/
/*DRAW_NODE..Draws a square at the node                                    */
/***************************************************************************/
draw_node(nodex,nodey)

int nodex,nodey;

{

   float startx,starty;          /* starting x and y of the grid */
   float x,y   ;                /* temp coord loc */

   /* compute the start of the grid */
   startx = ((XMAXSCREEN - MAXX * BOXWIDTH)/2.0)-145.0;
   starty = ((YMAXSCREEN - MAXY * BOXHEIGHT)/2.0)-20.0;
   /* compute the lowerleft of the box to fill */
   x = (startx + nodex*BOXWIDTH) + 3.0;
   y = (starty + nodey*BOXHEIGHT) + 3.0;
   /* determine the color of the node */
   determine_color(nodex,nodey);
   /* draw the filled rectangle */
   rectf(x,y,x+BOXWIDTH-3.0,y+BOXHEIGHT-3.0);
   /* if vehicle determine direction and draw */
   if(Wpt[nodex][nodey].state == AUV)
   draw_node_direction(nodex,nodey);

}

/***************************************************************************/
/*DETERMINE_COLOR..Determines the color of a node                          */
/***************************************************************************/
determine_color(nodex,nodey)

int nodex,nodey;

{

   if (Wpt[nodex][nodey].state == FREE)
     FREECOLOR
   if (Wpt[nodex][nodey].state == OBSTACLE)
     OBSTACLECOLOR
   if (Wpt[nodex][nodey].state == AUV)
     AUVCOLOR
   if (Wpt[nodex][nodey].state == ADJACENT)
     ADJACENTCOLOR
   if (Wpt[nodex][nodey].state == ACTIVE)
     ACTIVECOLOR
   if (Wpt[nodex][nodey].state == VISITED)
     VISITEDCOLOR
   if (Wpt[nodex][nodey].state == ASPATH)
     ASCOLOR

}

/***************************************************************************/
/*DETERMINE_VEHICLE_DIRECTION..Determines direction of vehicle             */
/***************************************************************************/
determine_vehicle_direction(dirx,diry)

int dirx,diry;
```

```
{

   float p,q;

   /* determine origin of direction pointer for vehicle movement */
   p = dirx+(BOXWIDTH/2.0)-2.0;
   q = diry+(BOXHEIGHT/2.0)-2.0;
   /* if auv draw direction */
   draw_node_direction(dirx,diry);

}

/**************************************************************************/
/*DRAW_NODE_DIRECTION..Draw the direction pointer of the node            */
/**************************************************************************/
draw_node_direction(nodex,nodey)

int nodex,nodey; /* node position being analyzed */

{

  float startx,starty;          /* starting x and y of the grid */
  float x,y   ;                 /* temp coord loc */
  float a,b;                    /* center of location of vehicle */

  /* compute the start of the grid */
  startx = ((XMAXSCREEN - MAXX * BOXWIDTH)/2.0)-145.0;
  starty = ((YMAXSCREEN - MAXY * BOXHEIGHT)/2.0)-20.0;
  /* compute the lowerleft of the box to fill */
  x = (startx + nodex*BOXWIDTH) + 2.0;
  y = (starty + nodey*BOXHEIGHT) + 2.0;
  /* determine origin of direction pointer for vehicle movement */
  a = x+(BOXWIDTH/2.0)-2.0;
  b = y+(BOXHEIGHT/2.0)-2.0;
  DIRECTIONCOLOR;
  linewidth(2);
  if(Wpt[nodex][nodey].dir == N)
  {
    move2(a,b);
    draw2(a,b+5.5);
  }
  if(Wpt[nodex][nodey].dir == NE)
  {
    move2(a,b);
    draw2(a+5.5,b+5.5);
  }
  if(Wpt[nodex][nodey].dir == E)
  {
    move2(a,b);
    draw2(a+5.5,b);
  }
  if(Wpt[nodex][nodey].dir == SE)
  {
    move2(a,b);
    draw2(a+5.5,b-5.5);
  }
  if(Wpt[nodex][nodey].dir == S)
  {
    move2(a,b);
    draw2(a,b-5.5);
  }
  if(Wpt[nodex][nodey].dir == SW)
  {
    move2(a,b);
    draw2(a-5.5,b-5.5);
  }
  if(Wpt[nodex][nodey].dir == W)
```

```
    {
      move2(a,b);
      draw2(a-5.5,b);
    }
    if(Wpt[nodex][nodey].dir == NW)
    {
      move2(a,b);
      draw2(a-5.5,b+5.5);
    }
    if(Wpt[nodex][nodey].dir == NODIR)
    {
      move2(a,b);
      draw2(a,b);
    }

}

/**************************************************************************/
/*MOUSEX_TO_BOX_POSITION..Determines what node mouse is pointing to      */
/**************************************************************************/
mousex_to_box_position(wx)

float wx;    /* mouse position */

{

  int mousex;    /* mouse position converted to grid position */
  float startx; /* lower left hand corner of area */

  startx = ((XMAXSCREEN - MAXX * BOXWIDTH)/2.0)-145.0;
  mousex = (wx - startx)/BOXWIDTH;
  return mousex;

}

/**************************************************************************/
/*MOUSEY_TO_BOX_POSITION..Determines what node mouse is pointing to      */
/**************************************************************************/
mousey_to_box_position(wy)

float wy;   /* mouse position */

{

  int mousey;    /* mouse position converted to grid position */
  float starty; /* lower left hand corner of area */

  starty = ((YMAXSCREEN - MAXY * BOXHEIGHT)/2.0)-20.0;
  mousey = (wy - starty)/BOXWIDTH;
  return mousey;

}

/**************************************************************************/
/*PUT_OBSTACLE..Draws an obstacle at indicated position and updates      */
/*the array                                                               */
/**************************************************************************/

put_obstacle(wx,wy)

float wx,wy; /* mouse position */

{

    int i,j;

    i = mousex_to_box_position(wx);
    j = mousey_to_box_position(wy);
    /* use grid position of mouse to update node contents */
    update_node_info(Wpt[i][j].grid.xpos,Wpt[i][j].grid.ypos);
    /* set the color of the node to be drawn */
    if(Wpt[i][j].state != OBSTACLE)
    {
      Wpt[i][j].state = OBSTACLE;
    }
    else
    {
      Wpt[i][j].state = FREE;
    }
    /* draw the specified node */
    draw_node(Wpt[i][j].grid.xpos,Wpt[i][j].grid.ypos);

}

/************************************************************************/
/*CHECK_NODE_INFO..Determins mouse hit and calls for info display at node*/
/************************************************************************/
check_node_info(wx,wy)

float wx,wy; /* mouse position */

{

    int i,j;

    i = mousex_to_box_position(wx);
    j = mousey_to_box_position(wy);
    /* use grid position of mouse to update node contents */
    update_node_info(Wpt[i][j].grid.xpos,Wpt[i][j].grid.ypos);

}

/************************************************************************/
/*UPDATE_NODE_INFO..Updates the display information for a node          */
/************************************************************************/
update_node_info(x,y,distravel,step,efratio)

int x,y; /* position of node of interest */
int step; /* number of steps taken by auv */
float distravel, /* cumulative distance traveled by auv */
      efratio;   /* efficiency ratio based on distance */

{

   char str1[20],
        str2[20],
        str3[20],
        str4[20],
        str5[20],
        str6[20];              /* screen data */
   static int mousex,mousey,
              mousestate,
              mousearea;       /* mouse info to screen */

   /* pull in information for indicated grid position */
   mousex = Wpt[x][y].grid.xpos;
   mousey = Wpt[x][y].grid.ypos;
   mousestate = Wpt[x][y].state;
   mousearea = Wpt[x][y].area;
   /* draw the print information background */
   RGBcolor(0,255,255);
   rectf(998,0,XMAXSCREEN,YMAXSCREEN);
   /* print information to screen */
   /* put up some help text */
   RGBcolor(255,0,0);
   cmov2i(10,1000);
```

```
    charstr("GRAPH SEARCH EVALUATION TOOL");
    RGBcolor(255,0,0);
    cmov2i(10,998);
    charstr("_____ ______ __________ ____");
    RGBcolor(0,0,0);
    cmov2i(10,985);
    charstr("              by Mark Compton");
    RGBcolor(0,0,255);
    cmov2i(1000,1000);
    charstr("Mouse Functions");
    cmov2i(1000,998);
    charstr("_____ _________");
    cmov2i(1000,985);
    charstr("Left:   Node Information");
    cmov2i(1000,970);
    charstr("Middle: Build Obstacle");
    cmov2i(1000,955);
    charstr("Right:  Menu");
    cmov2i(1000,935);
    charstr("Mouse Select Data");
    cmov2i(1000,933);
    charstr("_____ ______ ____");
    sprintf(str1, "x =  %d  y = %d",mousex,mousey);
    cmov2(1000,920);
    charstr(str1);
    sprintf(str2, "state = %d",mousestate);
    cmov2(1000,905);
    charstr(str2);
    sprintf(str3, "area = %d",mousearea);
    cmov2(1000,890);
    charstr(str3);
    RGBcolor(0,0,255);
    cmov2i(1000,100);
    charstr("Statistics");
    cmov2(1000,98);
    charstr("__________");
    sprintf(str4, "Graph steps = %d",step);
    cmov2(1000,80);
    charstr(str4);
    sprintf(str5, "Distance traveled = %f",distravel);
    cmov2(1000,65);
    charstr(str5);
    sprintf(str6, "Effective dist ratio = %f",efratio); /* min dist / dist */
    cmov2(1000,50);
    charstr(str6);

 }
```

APPENDIX D: THREE-DIMENSIONAL GRAPH SEARCH EVALUATION TOOL SOURCE CODE

```
/***************************************************************************
Title:  search3d.h
Author: Mark Compton
Course: Thesis
Date:   12 Mar 92

Description: This program builds a three dimensional search area designed to
             test graph type searches for theses work. The tool is called
             3-D GRAPH SEARCH EVALUATION TOOL.

Support: search3d.c (contains main menu for building tool)
         search_support3d.c (contains support routines for building tool)
***************************************************************************/

/*Preprocessing Directives************************************************/
#define BOXWIDTH  30.0  /* box width              */
#define BOXHEIGHT 30.0  /* box height             */
#define BOXDEPTH  30.0  /* box depth              */
#define MAXX 25         /* number of nodes in x direction */
#define MAXY 25         /* number of nodes in y direction */
#define MAXZ  9         /* number of nodes in z direction 9*/
#define SUBX 5          /* number of subareas in x direction */
#define SUBY 5          /* number of subareas in y direction */
#define SUBZ 3          /* number of zones in z direction 3*/
#define SUBXNODES 5     /* number of nodes in each subarea in x direction */
#define SUBYNODES 5     /* number of nodes in each subarea in y direction */
#define SUBZNODES 3     /* number of nodes in each zone in z direction 3*/
#define MAXSTRING 20    /* for file names */

#define ODD  1          /* defines if a value is odd */
#define EVEN 0          /* defines if a value is even */

#define SLOW 1000000          /* defines playback speeds */
#define MEDIUM 150000
#define FAST 15000

/* defines states of a node */
#define FREE     0      /* nothing at node */
#define OBSTACLE 1      /* obstacle at node */
#define AUV      2      /* vehicle (Autonomous Underwater Vehicle) */
#define ADJACENT 3      /* node adjacent to vehicle */
#define ACTIVE   4      /* detected, non-object, not visited */
#define VISITED  5      /* node has been previously visited */
#define ASPATH   6      /* local path selected by A-Star Search */

/* define colors of objects in search world */
#define FREECOLOR       RGBcolor(155,155,200);
#define OBSTACLECOLOR   RGBcolor(0,0,0);
#define AUVCOLOR        RGBcolor(0,0,255);
#define ADJACENTCOLOR   RGBcolor(110,45,0);
#define ACTIVECOLOR     RGBcolor(255,255,0);
#define VISITEDCOLOR    RGBcolor(255,255,255);
#define DIRECTIONCOLOR  RGBcolor(255,255,0);
#define ASCOLOR         RGBcolor(255,0,0);

#define STOREOBSTACLES 10       /* menu pick for storing obstacle data */
#define RETRIEVEOBSTACLES 20    /* menu pick for retrieving obstacle data */
#define RETRIEVEMISSION 30      /* menu pick for vehicle track */
```

```
#define PLAYBACKMISSION 40      /* menu pick for mission playback */
#define PAUSEMISSION 42         /* menu pick to pause mission playback */
#define RESUMEMISSION 44        /* menu pick to resume mission playback */
#define CLEARAREA 50            /* menu pick to exit program */
#define PLAYBACKSPEEDSLOW 60    /* menu pick for playback speeds */
#define PLAYBACKSPEEDMEDIUM 61  /* menu pick for playback speeds */
#define PLAYBACKSPEEDFAST 62    /* menu pick for playback speeds */
#define EXIT         70         /* menu pick to exit program */

/* defines direction of vehicle */
#define NODIR 0
#define N     1
#define NE    2
#define E     3
#define SE    4
#define S     5
#define SW    6
#define W     7
#define NW    8
#define UNODIR 9
#define UN     10
#define UNE    11
#define UE     12
#define USE    13
#define US     14
#define USW    15
#define UW     16
#define UNW    17
#define DNODIR 18
#define DN     19
#define DNE    20
#define DE     21
#define DSE    22
#define DS     23
#define DSW    24
#define DW     25
#define DNW    26

/*Globals***************************************************************/
char outnode[MAXSTRING];      /* output file for nodes */
FILE *obstacleofp;            /* pointer for sending obstacles to file */
FILE *obstacleifp;            /* pointer for receiving obstacles from file */
FILE *missionifp;             /* pointer for receiving mission from file */
char inobstacle[MAXSTRING];     /* input file name for obstacles */
char inmission[MAXSTRING];      /* input file name for mission */
int mouse_select = FALSE;       /* toggle for mouse selections */
int left_mouse_select = FALSE;  /* toggle for mouse selections */
int playthemission = FALSE;     /* flag for playback of mission */
int current_level;              /* indicates current level for display */
int fullinitialize = TRUE;      /* reinitializes full world map */
int playbackspeed = MEDIUM;     /* initializes playback speed *

/*Structures************************************************************/
struct location           /* basic structure for location of an entity */
{
  int xpos,ypos,zpos;     /* grid coordinates */
};

typedef struct location Location;

struct node               /* basic structure of a search area node */
{
  Location grid;          /* grid coordinates of node */
  int zone;               /* zone node is located in */
  double x,y,z;           /* world position of node */
  int dir;                /* direction node is looking */
  int state;              /* status of node */
  int area;               /* area of node in a level */
  int visited;          /* node previously visited by vehicle flag */
};
  typedef struct node Node;

Node Wpt[MAXX][MAXY][MAXZ]; /* 3-D array for storing search area nodes */

struct missiondata
{
  char info;                         /* type of information for display */
  Location grid;                     /* grid coordinates of node */
  int dir;                           /* direction vehicle is looking */
  int state;                         /* status of node */
  struct missiondata *nextmissiondata; /* points to next mission data */
};
  typedef struct missiondata Missiondata;
  typedef Missiondata *MISSIONLINK;  /* pointer to Missiondata */
  MISSIONLINK head;                  /* points to head of list */
  MISSIONLINK current;               /* points to item in mission list */
  MISSIONLINK prev;                  /* points to previous in mission list */

/*************************************************************************
Title:  search3d.c
Author: Mark Compton
Course: Thesis
Date:   12 Mar 92

Description: This program builds a two dimensional search area designed to
             test graph type searches for theses work.
*************************************************************************/

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "gl.h"
#include "device.h"
#include "search3d.h"
#include "search_support3d.c"

main ()

{

  char  response = 'n';
  char  answer   = 'n';
  char  acknowledgement = 'n';
  int mainmenu;
  short value;      /* the value from the Event Queue */
  float wx,wy;      /* world coord location of the mouse */
  float wz;         /* world coord location of node level */
  long  i,j;        /* counters for send to file */
  int hititem;      /* variable holding hit name */

    /* Setup for retrieving mission data */
    printf("\n\nMission data may be retrieved from a file ");
    printf("\nthen displayed by selecting Retrieve Mission in");
    printf("\nthe menu.\n\n ");
    printf("Will you be retrieving mission data from a file?  ");
    scanf("%c",&response);
    if (response == 'y' || response == 'Y')
    {
      printf("\n\nPlease enter the mission input file name: \n");
      scanf("%s",inmission);
    }
    /* Setup for retrieving obstacle data */
    scanf("%c",&answer); /* hack to clear carrage return from buffer */
    printf("\nObstacle data may be retrieved from a file ");
    printf("\nthen displayed by selecting Retrieve Obstacle in");
    printf("\nthe menu.\n\n ");
```

```
 printf("Will you be retrieving obstacle data from a file?  ");
 scanf("%c",&answer);
 if (answer == 'y' || answer == 'Y')
 {
   printf("\n\nPlease enter the obstacle input file name: \n");
   scanf("%s",inobstacle);
 }
/* initialize the graphics system */
initialize();
/* make the popup menus */
mainmenu = makethemenus();
/* set the world coordinate system */
ortho2(0.0,(float)XMAXSCREEN,0.0,(float)YMAXSCREEN);
/* stay inside this display loop until the
   Window Manager terminates us...
*/
/* first time thru for drawing area */
fullinitialize = TRUE;
drawthearea();
while(TRUE)
{
   /* is there anything on the Event Queue? */
   while(qtest() != 0)
   {
      switch(qread(&value))
      {
         case REDRAW:
                     reshapeviewport();
                     break;
         case LEFTMOUSE:
                     /* if the button has been depressed */
                     if(value == 1)
                     {
                       /* compute the world coordinate of the mouse */
                        wx = getvaluator(MOUSEX);
                        wy = getvaluator(MOUSEY);
                        wz = current_level;
                        /* left_mouse_select = TRUE; */
                        if (wx < 1000.0)
                        {
                          check_node_info(wx,wy,wz);
                        }
                        else
                        {
                          select_level(wx,wy,wz);
                        }
                     }
                     break;
         case MIDDLEMOUSE:
                     /* if the button has been depressed */
                     if(value == 1)
                     {
                        /* compute the world coordinate of the mouse */
                        wx = getvaluator(MOUSEX);
                        wy = getvaluator(MOUSEY);
                        wz = current_level;
                        /* mouse_select = TRUE; */
                        if (wx < 1000.0)
                        {
                          put_obstacle(wx,wy,wz);
                        }
                        else
                        {
                          select_level(wx,wy,wz);
                        }
                     }
                     break;
         case MENUBUTTON:
```

```
                    if(value == 1)
                    {
                       hititem = dopup(mainmenu);

                       processmenuhit(hititem);
                    }
                    break;

                 default:
                             break;
              } /* end switch */
           } /* endif there was something on the queue */
           if (playthemission == TRUE)
           {
              playbackmission();
           }
        } /* end while loop */

}

/*************************************************************************
Title:  search_support3d.c
Author: Mark Compton
Course: Thesis
Date:   12 Mar 92

Description: This program contains routines which support the building of
             a two dimensional graph search evaluation tool.
*************************************************************************/

#include <stdio.h>
#include <stdlib.h>
#include <math.h>

/************************************************************************/
/*INITIALIZE..Establish IRIS graphics preferences                       */
/************************************************************************/
initialize()

{

   /* set up a preferred size for the window */
   prefsize(XMAXSCREEN+1,YMAXSCREEN+1);
   /* set up a preferred position for the window */
   prefposition(0,XMAXSCREEN,0,YMAXSCREEN);
   /* open a window for the program */
   winopen("pick");
   /* put a title on the window */
   wintitle("Pick -- Middle Mouse is the Pick ");
   /* set color type */
   RGBmode();
   /* put the IRIS into double buffer mode */
   /* doublebuffer(); */
   singlebuffer();
   /* configure the IRIS (means use the above command settings) */
   gconfig();
   /* queue the redraw device */
   qdevice(REDRAW);
   /* queue the menubutton */
   qdevice(MENUBUTTON);
   /* queue the left mouse button */
   qdevice(LEFTMOUSE);
   /* queue the middle mouse button */
   qdevice(MIDDLEMOUSE);
   /* initialize the current level */
   current_level = 0;
   /* select flat shading */
   /* only works on the 4D*/
/* #ifdef FLAT
   shademodel(FLAT);
   #endif FLAT */

}

/*************************************************************************/
/*MAKETHEMENUS..Performs all the menu construction call                  */
/*************************************************************************/
int makethemenus()

{

   int topmenu;   /* top level menu's name */

   /* build the top level menu */
   topmenu = defpup("Search Menu %t | Store Obstacles %x10 \
                                    | Retrieve Obstacles %x20 \
                                    | Retrieve Mission Data %x30 \
                                    | Playback Mission %x40 \
                                    | Pause Mission Playback %x42 \
                                    | Resume Mission Playback %x44 \
                                    | Clear Area %x50 \
                                    | Playback Slow %x60 \
                                    | Playback Medium %x61 \
                                    | Playback Fast %x62 \
                                    | Exit %x70 ");
   /* return the name of this menu */
   return(topmenu);

}


/*************************************************************************/
/*PROCESSMENUHIT..Performs all the menu actions                          */
/*************************************************************************/
processmenuhit(hititem)

int hititem;   /* item hit on the popup menus */

{

   switch(hititem)
   {
         case STOREOBSTACLES:
                    storedata();
                    break;
         case RETRIEVEOBSTACLES:
                    retrieveobstacledata();
                    break;
         case RETRIEVEMISSION:
                    retrievemission();
                    break;
         case PLAYBACKMISSION:
                    playthemission = TRUE;
                    current = head;
                    prev = head;
                    /* playbackmission(); */
                    break;
         case PAUSEMISSION:
                    playthemission = FALSE;
                    break;
         case RESUMEMISSION:
                    playthemission = TRUE;
                    break;
         case CLEARAREA:
```

```
                      fullinitialize = TRUE;
                      drawthearea();
                      playthemission = FALSE;
                      break;
         case PLAYBACKSPEEDSLOW:
                      playbackspeed = SLOW;
                      break;
         case PLAYBACKSPEEDMEDIUM:
                      playbackspeed = MEDIUM;
                      break;
         case PLAYBACKSPEEDFAST:
                      playbackspeed = FAST;
                      break;
         case EXIT:
                      exit(0);
                      break;
         default:
                      break;
      } /* end switch */

}

/*************************************************************************/
/*DRAWTHEAREA..Draws the graph search environment                        */
/*************************************************************************/
drawthearea()

{

   float startx,starty;          /* starting x and y of the grid */
   float x,y,c,d;                /* temp coord loc */
   long i,j,a,b;                 /* temp loop counters */

   /* draw the background color */
   RGBcolor(0,255,255);
   clear();
   /* put up some help text */
   RGBcolor(255,0,0);
   cmov2i(10,1000);
   charstr("3-D GRAPH SEARCH EVALUATION TOOL");
   RGBcolor(255,0,0);
   cmov2i(10,998);
   charstr("___ _____ ______ __________ ____");
   RGBcolor(0,0,0);
   cmov2i(10,985);
   charstr("                 by Mark Compton");
   RGBcolor(0,0,255);
   cmov2i(1000,1000);
   charstr("Mouse Functions");
   cmov2i(1000,998);
   charstr("_____ _________");
   cmov2i(1000,985);
   charstr("Left:   Node Information");
   cmov2i(1000,970);
   charstr("Middle: Build Obstacle");
   cmov2i(1000,955);
   charstr("Right:  Menu");
   cmov2i(1000,935);
   charstr("Mouse Select Data");
   cmov2i(1000,933);
   charstr("_____ ______ ____");
   cmov2(1000,920);
   charstr("x =      y =   z =      ");
   cmov2(1000,905);
   charstr("state =   ");
   cmov2(1000,890);
   charstr("area =   ");
   cmov2(1000,850);         /* menu for selecting and indicating level */
```

```
charstr("Select Level");
cmov2(1000,848);
charstr("______ ______");
cmov2(1000,820);
RGBcolor(0,0,0);
charstr("0");
rectf(1020,800,1060,840);
cmov2(1000,770);
RGBcolor(255,0,0);
charstr("1");
rectf(1020,750,1060,790);
cmov2(1000,720);
RGBcolor(0,255,0);
charstr("2");
rectf(1020,700,1060,740);
RGBcolor(0,0,0);
cmov2(1000,691);
charstr("------------");
cmov2(1000,670);
RGBcolor(0,0,255);
charstr("3");
rectf(1020,650,1060,690);
cmov2(1000,620);
RGBcolor(255,255,255);
charstr("4");
rectf(1020,600,1060,640);
cmov2(1000,570);
RGBcolor(255,0,255);
charstr("5");
rectf(1020,550,1060,590);
RGBcolor(0,0,0);
cmov2(1000,541);
charstr("------------");
cmov2(1000,520);
RGBcolor(0,124,124);
charstr("6");
rectf(1020,500,1060,540);
cmov2(1000,470);
RGBcolor(255,124,124);
charstr("7");
rectf(1020,450,1060,490);
cmov2(1000,420);
RGBcolor(255,75,0);
charstr("8");
rectf(1020,400,1060,440);
pointtolevel();  /* circle marks the current level shown */
/* compute the start of the grid */
startx = ((XMAXSCREEN - MAXX * BOXWIDTH)/2.0)-145.0;
starty = ((YMAXSCREEN - MAXY * BOXHEIGHT)/2.0)-20.0;
/* draw background subareas */
  for(a=0; a < MAXY/SUBYNODES; ++a)
  {
    for(b=0; b < MAXX/SUBXNODES; ++b)
    {
       /* compute the color */
      if (odd(a) == 0)
      {
        if (odd(b) == 0)
        {
        RGBcolor(0,255,0);
        }
        if (odd(b) == 1)
        {
        RGBcolor(255,0,255);
        }
       }
      if (odd(a) == 1)
      {
           if (odd(b) == 0)
           {
           RGBcolor(255,0,255);
           }
           if (odd(b) == 1)
           {
           RGBcolor(0,255,0);
           }
          }
          /* compute the lowerleft of the box to fill */
          c = (startx + a*BOXWIDTH*SUBXNODES);
          d = (starty + b*BOXHEIGHT*SUBYNODES);
          /* draw the filled rectangles */
          rectf(c,d,c+BOXWIDTH*SUBXNODES
                   ,d+BOXHEIGHT*SUBYNODES);
       }
     }
   colorlevelboarder();
   linewidth(10);
   rect(startx-2.5,starty-2.5,
        startx+5.0+(BOXWIDTH*MAXX),starty+5.0+(BOXHEIGHT*MAXY));
   if (fullinitialize == TRUE)  /* if first time thru or clear area selected */
     initialize_the_area();     /* resets addresses of grids */
   /* draw all the nodes */
   for(j=0; j < MAXY; j=j+1)
   {
      for(i=0; i < MAXX; i=i+1)
      {
         /* compute the lowerleft of the box to fill */
         x = (startx + i*BOXWIDTH) + 4.0;
         y = (starty + j*BOXHEIGHT) + 4.0;
         /* determine color of node */
         if  (Wpt[i][j][current_level].state == OBSTACLE)
           OBSTACLECOLOR;
         if  (Wpt[i][j][current_level].state == FREE)
           FREECOLOR;
         if(Wpt[i][j][current_level].state == AUV)
           AUVCOLOR;
         if(Wpt[i][j][current_level].state == ADJACENT)
           ADJACENTCOLOR;
         if(Wpt[i][j][current_level].state == ACTIVE)
           ACTIVECOLOR;
         if(Wpt[i][j][current_level].state == VISITED)
           VISITEDCOLOR;
         if(Wpt[i][j][current_level].state == ASPATH)
           ASCOLOR;
         /* draw the filled rectangles */
         rectf(x,y,x+BOXWIDTH-4.0,y+BOXHEIGHT-4.0);
      }
   }
   fullinitialize = FALSE; /* allows world memory to remain intact */

}

/***************************************************************************/
/*ODD..Determines if a value is odd or even                                */
/***************************************************************************/
odd(value)

  int value;

{

  if (value % 2 == 0)
    return EVEN;
  else
    return ODD;
```

```
}

/*************************************************************************/
/*EVEN..Determines if a value is even  or odd                            */
/*************************************************************************/
int even(value)

  int value;

{

  if (value % 2 == 0)
    return ODD;
  else
    return EVEN;

}

/*************************************************************************/
/*INSIDE..determines if (x,y) is inside the box defined by the           */
/*coordinates (xmin,ymin)-(xmax,ymax)                                    */
/*NOT CURRENTLY USED..DELETE BEFORE FINAL PRODUCT                        */
/*************************************************************************/
int inside(x,y,xmin,ymin,xmax,ymax)

float x,y;   /* location of the cursor */

float xmin,ymin,xmax,ymax;   /* bounding box to check if cursor is inside */

{

   if((xmin <= x) && (x <= xmax) && (ymin <= y) && (y <= ymax))
   {
      return(TRUE);
   }
   else
   {
      return(FALSE);
   }

}

/*************************************************************************/
/*STOREDATA..Stores node data to file if requested                       */
/*************************************************************************/
storedata()

{

  int i,j,k;

  obstacleofp = fopen("obstacle_file","w");
  for(k=0; k < MAXZ; k=k+1)
  {
    for(i=0; i < MAXY; i=i+1)
    {
      for(j=0; j < MAXX; j=j+1)
      {
        if(Wpt[i][j][k].state == OBSTACLE)
        {
        fprintf(obstacleofp,"%d %d %d \n",i,j,k);
        }
      }
    }
  }

  fclose(obstacleofp);
```

```
}

/****************************************************************************/
/*RETRIEVEOBSTACLEDATA..Retrieves obstacle data from file if requested   */
/****************************************************************************/
retrieveobstacledata()

{

  int x,y,z;

  obstacleifp = fopen(inobstacle,"r");
  while(fscanf(obstacleifp, "%d%d%d", &x,&y,&z) != EOF)
  {
    Wpt[x][y][z].state = OBSTACLE;
    if (current_level == z)

draw_node(Wpt[x][y][z].grid.xpos,Wpt[x][y][z].grid.ypos,Wpt[x][y][z].grid.zpos);
  }
  fclose(obstacleifp);
  drawthearea();

}

/****************************************************************************/
/*RETRIEVEMISSION..Retrieves mission data from file if requested         */
/*Stores mission data in linked list for future use                      */
/****************************************************************************/
retrievemission()

{

  int x,y,z,nodedir,nodestate;
  char w; /* w indicates type of line to follow */
  MISSIONLINK currentptr,previousptr;

  /* draw the print information background */
  RGBcolor(0,0,0);
  rectf(270,420,730,580);
  RGBcolor(255,0,0);
  rectf(300,450,700,550);
  RGBcolor(0,0,0);
  cmov2i(400,500);
  charstr("STANDBY----LOADING MISSION");
  missionifp = fopen(inmission,"r");
  fscanf(missionifp, "%c%d%d%d%d%d",&w,&x,&y,&z,&nodedir,&nodestate);
    head = malloc(sizeof(Missiondata));
    head -> info      = w;
    head -> grid.xpos = x;
    head -> grid.ypos = y;
    head -> grid.zpos = z;
    head -> dir       = nodedir;
    head -> state     = nodestate;
    previousptr = head;
  while (fscanf(missionifp, "%c%d%d%d%d%d",&w,&x,&y,&z,&nodedir,&nodestate)
                                                                    != EOF)
  {
    currentptr = malloc(sizeof(Missiondata));
    currentptr -> info      = w;
    currentptr -> grid.xpos = x;
    currentptr -> grid.ypos = y;
    currentptr -> grid.zpos = z;
    currentptr -> dir       = nodedir;
    currentptr -> state     = nodestate;
    previousptr -> nextmissiondata = currentptr;
    previousptr = currentptr;
  }
  fclose(missionifp);
  drawthearea(); /* get rid of message */

}

/*************************************************************************/
/*KILL_TIME..Kills processor time for visual purposes in drawing        */
/*************************************************************************/
kill_time()

{

int a,b,c;

  for(a=0;a<playbackspeed;++a)
    b = 10000000;
    c = a*b;

}

/*************************************************************************/
/*PLAYBACKMISSION..Playback of mission data                             */
/*************************************************************************/
playbackmission()

{

  if(current == NULL)
    playthemission = FALSE;
  else
  {
    if (current != NULL)
    {
      if(current->info == '%') /* world information to follow */
      {
        /* assign input state to node */
        Wpt[current->grid.xpos][current->grid.ypos][current->grid.zpos].state =
                                                            current->state;
        if(current->grid.zpos == current_level)
        {
          draw_node(Wpt[current->grid.xpos][current->grid.ypos][current-
>grid.zpos].grid.xpos,
                    Wpt[current->grid.xpos][current->grid.ypos][current-
>grid.zpos].grid.ypos,
                    Wpt[current->grid.xpos][current->grid.ypos][current-
>grid.zpos].grid.zpos);
        }
      }
      if(current->info == '$') /* search path information to follow */
      {
        /* assign input state to node */
        Wpt[current->grid.xpos][current->grid.ypos][current->grid.zpos].state =
                                                            current->state;
        if(current->state == 2 || current->state == 5 || current->state == 6 ||
          (current->state == 4 && current->grid.zpos == current_level))
        {
          draw_node(Wpt[current->grid.xpos][current->grid.ypos][current-
>grid.zpos].grid.xpos,
                    Wpt[current->grid.xpos][current->grid.ypos][current-
>grid.zpos].grid.ypos,
                    Wpt[current->grid.xpos][current->grid.ypos][current-
>grid.zpos].grid.zpos);
          kill_time(); /* used for better visual presentation */
        }
      }
      prev = current;
      current = current->nextmissiondata;
      /* free(prev); use only if desire to delete list */
    }
  }

}

/***************************************************************************/
/*SUBAREA..Determines which subarea a node is in on each level             */
/***************************************************************************/
int subarea(a,b,c)

int a,b,c; /* feed in the grid coordinates */

{

  int i,j,k,row,column,level.subarea;

  for (k = 1; k <= SUBZ; ++k)                /* step thru subarea levels */
  {
    for (i = 1; i <= SUBY; ++i)              /* step thru subarea rows */
    {
      for (j = 1; j <= SUBX; ++j)            /* step thru subarea columns */
      {
        if (c >= (k-1)*SUBZNODES && c < k*SUBZNODES)
        {
          level = k;                         /* subarea level */
        }
        if (b >= (i-1)*SUBYNODES && b < i*SUBYNODES)
        {
          row = i;                           /* subarea row */
        }
        if (a >= (j-1)*SUBXNODES && a < j*SUBXNODES)
        {
          column = j;                        /* subarea column */
        }
      }
    }
  }
  if (odd(level))  /* increase from right to left in odd rows */
  {
    if (odd(row))
    {
      subarea = (((row-1)*(SUBZ*SUBX))+column)+((level-1)*SUBX);
    }
    if (even(row))
    {
      subarea = (((row*SUBZ*SUBX)+1)-column)-((level-1)*SUBX);
    }
  }
  if (even(level)) /* decrease from right to left in odd rows */
  {
    if (odd(row))
    {
      subarea = (((row*SUBZ*SUBX)+1)-column)-((level-1)*SUBX);
    }
    if (even(row))
    {
      subarea = ((((row-1)*SUBZ*SUBX))+column)+((level-1)*SUBX);
    }
  }
  return subarea;

}

/***************************************************************************/
/*DRAW_NODE..Draws a square at the node                                    */
/***************************************************************************/
draw_node(nodex,nodey,nodez)
```

```
int nodex,nodey,nodez;

{

   float startx,starty;          /* starting x and y of the grid */

   float x,y   ;                 /* temp coord loc */

   if(nodez != current_level)
   {
     current_level = nodez;
     drawthearea();
   }
   /* compute the start of the grid */
   startx = ((XMAXSCREEN - MAXX * BOXWIDTH)/2.0)-145.0;
   starty = ((YMAXSCREEN - MAXY * BOXHEIGHT)/2.0)-20.0;
   /* compute the lowerleft of the box to fill */
   x = (startx + nodex*BOXWIDTH) + 4.0;
   y = (starty + nodey*BOXHEIGHT) + 4.0;
   /* determine the color of the node */
   determine_color(nodex,nodey,nodez);
   /* draw the filled rectangle */
   rectf(x,y,x+BOXWIDTH-4.0,y+BOXHEIGHT-4.0);

}

/**************************************************************************/
/*DETERMINE_COLOR..Determines the color of a node                         */
/**************************************************************************/
determine_color(nodex,nodey,nodez)

int nodex,nodey,nodez;

{

   if (Wpt[nodex][nodey][nodez].state == FREE)
     FREECOLOR
   if (Wpt[nodex][nodey][nodez].state == OBSTACLE)
     OBSTACLECOLOR
   if (Wpt[nodex][nodey][nodez].state == AUV)
     AUVCOLOR
   if (Wpt[nodex][nodey][nodez].state == ADJACENT)
     ADJACENTCOLOR
   if (Wpt[nodex][nodey][nodez].state == ACTIVE)
     ACTIVECOLOR
   if (Wpt[nodex][nodey][nodez].state == VISITED)
     VISITEDCOLOR
   if (Wpt[nodex][nodey][nodez].state == ASPATH)
     ASCOLOR

}

/**************************************************************************/
/*MOUSEX_TO_BOX_POSITION..Determines what node mouse is pointing to       */
/**************************************************************************/
mousex_to_box_position(wx)

float wx;    /* mouse position */

{

  int mousex;    /* mouse position converted to grid position */
  float startx; /* lower left hand corner of area */

  startx = ((XMAXSCREEN - MAXX * BOXWIDTH)/2.0)-145.0;
  mousex = (wx - startx)/BOXWIDTH;
  return mousex;
```

```
}

/****************************************************************************/
/*MOUSEY_TO_BOX_POSITION..Determines what node mouse is pointing to       */
/****************************************************************************/
mousey_to_box_position(wy)

float wy;  /* mouse position */

{

  int mousey;   /* mouse position converted to grid position */
  float starty; /* lower left hand corner of area */

  starty = ((YMAXSCREEN - MAXY * BOXHEIGHT)/2.0)-20.0;
  mousey = (wy - starty)/BOXWIDTH;
  return mousey;

}

/****************************************************************************/
/*PUT_OBSTACLE..Draws an obstacle at indicated position and updates        */
/*the array                                                                 */
/****************************************************************************/

put_obstacle(wx,wy,wz)

float wx,wy,wz; /* mouse position */

{

    int i,j,k;

    i = mousex_to_box_position(wx);
    j = mousey_to_box_position(wy);
    k = wz;
    /* use grid position of mouse to update node contents */
    update_node_info(i,j,k);
    /* set the color of the node to be drawn */
    if(Wpt[i][j][k].state != OBSTACLE)
    {
      Wpt[i][j][k].state = OBSTACLE;
    }
    else
    {
      Wpt[i][j][k].state = FREE;
    }
    /* draw the specified node */
    draw_node(i,j,k);

}

/****************************************************************************/
/*CHECK_NODE_INFO..Determins mouse hit and calls for info display at node*/
/****************************************************************************/
check_node_info(wx,wy)

float wx,wy; /* mouse position */

{

    int i,j,k;

    i = mousex_to_box_position(wx);
    j = mousey_to_box_position(wy);
    k = current_level;
    /* use grid position of mouse to update node contents */
    update_node_info(i,j,k);
```

```
}

/***************************************************************************/
/*UPDATE_NODE_INFO..Updates the display information for a node            */
/***************************************************************************/
update_node_info(x,y,z)

int x,y,z; /* position of node of interest */

{

   char str1[20],
        str2[20],
        str3[20];                /* screen data */
   static int mousex,mousey,mousez,
              mousestate,
              mousearea;         /* mouse info to screen */

   /* pull in information for indicated grid position */
   mousex = Wpt[x][y][z].grid.xpos;
   mousey = Wpt[x][y][z].grid.ypos;
   mousez = Wpt[x][y][z].grid.zpos;
   mousestate = Wpt[x][y][z].state;
   mousearea = Wpt[x][y][z].area;
   /* draw the print information background */
   RGBcolor(0,255,255);
   rectf(998,0,XMAXSCREEN,YMAXSCREEN);
   /* print information to screen */
   /* put up some help text */
   RGBcolor(255,0,0);
   cmov2i(10,1000);
   charstr("3-D GRAPH SEARCH EVALUATION TOOL");
   RGBcolor(255,0,0);
   cmov2i(10,998);
   charstr("___ _____ ______ __________ ____");
   RGBcolor(0,0,0);
   cmov2i(10,985);
   charstr("                 by Mark Compton");
   RGBcolor(0,0,255);
   cmov2i(1000,1000);
   charstr("Mouse Functions");
   cmov2i(1000,998);
   charstr("_______________");
   cmov2i(1000,985);
   charstr("Left:   Node Information");
   cmov2i(1000,970);
   charstr("Middle: Build Obstacle");
   cmov2i(1000,955);
   charstr("Right:  Menu");
   cmov2i(1000,935);
   charstr("Mouse Select Data");
   cmov2i(1000,933);
   charstr("_____ ______ ____");
   sprintf(str1, "x =  %d  y = %d  z = %d",mousex,mousey,mousez);
   cmov2(1000,920);
   charstr(str1);
   sprintf(str2, "state = %d",mousestate);
   cmov2(1000,905);
   charstr(str2);
   sprintf(str3, "area = %d",mousearea);
   cmov2(1000,890);
   charstr(str3);
   cmov2(1000,850);
   charstr("Select Level");
   cmov2(1000,848);
   charstr("______ _____");
   cmov2(1000,820);
```

```
    RGBcolor(0,0,0);
    charstr("0");
    rectf(1020,800,1060,840);
    cmov2(1000,770);
    RGBcolor(255,0,0);
    charstr("1");
    rectf(1020,750,1060,790);
    cmov2(1000,720);
    RGBcolor(0,255,0);
    charstr("2");
    rectf(1020,700,1060,740);
    RGBcolor(0,0,0);
    cmov2(1000,691);
    charstr("------------");
    cmov2(1000,670);
    RGBcolor(0,0,255);
    charstr("3");
    rectf(1020,650,1060,690);
    cmov2(1000,620);
    RGBcolor(255,255,255);
    charstr("4");
    rectf(1020,600,1060,640);
    cmov2(1000,570);
    RGBcolor(255,0,255);
    charstr("5");
    rectf(1020,550,1060,590);
    RGBcolor(0,0,0);
    cmov2(1000,541);
    charstr("------------");
    cmov2(1000,520);
    RGBcolor(0,124,124);
    charstr("6");
    rectf(1020,500,1060,540);
    cmov2(1000,470);
    RGBcolor(255,124,124);
    charstr("7");
    rectf(1020,450,1060,490);
    cmov2(1000,420);
    RGBcolor(255,75,0);
    charstr("8");
    rectf(1020,400,1060,440);
    pointtolevel();

}

/****************************************************************************/
/*COLORLEVELBOARDER..Updates the boarder color for each level              */
/****************************************************************************/
colorlevelboarder()

{

  if (current_level == 0)
   RGBcolor(0,0,0);
  if (current_level == 1)
   RGBcolor(255,0,0);
  if (current_level == 2)
   RGBcolor(0,255,0);
  if (current_level == 3)
   RGBcolor(0,0,255);
  if (current_level == 4)
   RGBcolor(255,255,255);
  if (current_level == 5)
   RGBcolor(255,0,255);
  if (current_level == 6)
   RGBcolor(0,124,124);
  if (current_level == 7)
   RGBcolor(255,124,124);
  if (current_level == 8)
   RGBcolor(255,75,0);

}

/**************************************************************************/
/*POINTTOLEVEL..Points to the current level                              */
/**************************************************************************/
pointtolevel()

{

  int ystart;  /* where to start drawing */

  colorlevelboarder();
  ystart = 820 - (current_level * 50);
  circf(1090,ystart,10);

}

/**************************************************************************/
/*SELECTLEVEL..Selects the desired level                                 */
/**************************************************************************/
select_level(i,j)

  float i,j; /* position of mouse */

{
      /* see if it is to select new level */
      if (i >= 1020.0 && i <= 1060.0)
      {
        if (j >= 800.0 && j <= 840.0)
        {
          current_level = 0;
          drawthearea();
        }
        if (j >= 750.0 && j <= 790.0)
        {
          current_level = 1;
          drawthearea();
        }
        if (j >= 700.0 && j <= 740.0)
        {
          current_level = 2;
          drawthearea();
        }
        if (j >= 650.0 && j <= 690.0)
        {
          current_level = 3;
          drawthearea();
        }
        if (j >= 600.0 && j <= 640.0)
        {
          current_level = 4;
          drawthearea();
        }
        if (j >= 550.0 && j <= 590.0)
        {
          current_level = 5;
          drawthearea();
        }
        if (j >= 500.0 && j <= 540.0)
        {
          current_level = 6;
          drawthearea();
        }
        if (j >= 450.0 && j <= 490.0)
        {
          current_level = 7;
          drawthearea();
        }
        if (j >= 400.0 && j <= 440.0)
        {
          current_level = 8;
          drawthearea();
        }
      }

}

/***************************************************************************/
/*INITIALIZE_THE_AREA..Reinitializes the entire area                      */
/***************************************************************************/
initialize_the_area()

{

  int i,j,k;

  for(k=0; k< MAXZ; k=k+1)
  {
    for(j=0; j < MAXY; j=j+1)
    {
      for(i=0; i < MAXX; i=i+1)
      {
         /* fill in Node structure */
           Wpt[i][j][k].x = i;
           Wpt[i][j][k].y = j;
           Wpt[i][j][k].z = k;
           Wpt[i][j][k].state = FREE;
           Wpt[i][j][k].grid.xpos = i;
           Wpt[i][j][k].grid.ypos = j;
           Wpt[i][j][k].grid.zpos = k;
           Wpt[i][j][k].area = subarea(i,j,k);
      }
    }
  }

}
```

APPENDIX E: AUV SIMULATOR GUIDANCE MODULE SOURCE CODE

```
/**************************************************************************
Title:  guidance.h
Author: Mark Compton
Course: Thesis
Date:   09 March 92

Description:
This program analyzes the path of a point robot as it maneuvers
from waypoint to waypoint. It then outputs this information to
a graph file where it can be printed or to a NPS AUV Integrated
Graphic Simulator file for playback.

The objectives are as follows:
Case 1: Determine path of robot based on initial configuration,
        speed, turn direction, turn rate and travel time.
Case 2: Determine path of robot direct from start point to first
        available waypoint, then from waypoint to waypoint.

**************************************************************************/
/*Preprocessing Directives************************************************/
#define PI             3.141592653589793
#define MAXSTRING     20
#define SUCCESS        2
#define FAILURE        0
#define FATAL          1
#define FALSE          0
#define TRUE           1
#define NONFATAL       0
#define UNDEFINED     -1

/*Global Declarations*****************************************************/
int     verbose = FALSE;           /*trouble shooting boolean*/
int     count = 0;                 /*track iterations of Dr inputs*/
int     desired_case = 0;
int     last_visited_wpt = -1;     /*eliminates reanalysis of visited wpts*/
int     total_wpts = 0;
int     total_vertices = 0;
char    outdrxy[MAXSTRING];
char    outdrxz[MAXSTRING];
char    outsim1[MAXSTRING];
char    outsim2[MAXSTRING];
char    outwptxy[MAXSTRING];
char    outwptxz[MAXSTRING];
char    inwpt[MAXSTRING];
int     send_to_file = 0;
FILE    *drxyofp,*drxzofp,*sim1ofp,*sim2ofp,
        *wptxyofp,*wptxzofp,*wptifp;

/*Structures**************************************************************/
struct point
{
  double x,y,z,azimuth,elevation;
};
  typedef struct point Point;

struct config
{
  double x, y, z, azimuth, elevation;
};
  typedef struct config Config;
```

```
Config Dr[100000];
Config Wpt[10000];

/*Functions*****************************************************************/
extern double max_val();
extern double min_val();
extern double rad_to_deg();
extern double deg_to_rad();
extern double normalize();
extern double relazimuth();
extern double relelev();
extern int precede();
extern Config compute_new_configuration();
extern void output_xy_dr_points();
extern void output_xz_dr_points();
extern void output_siml_points();
extern void output_ztime_points();
extern void enter_waypoints();
extern void retrieve_waypoints();
extern int determine_next_waypoint();
extern void proceed_along_path();
extern int introduction();
extern void case_1();
extern void case_2();


/*Main Program**************************************************************/

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <string.h>
#include "guidance.h"

main ()
{
  char   answer   = 'y';
  char   response = 'n';
  char   outtemp[MAXSTRING];
  Config *config_a,*config_b,*config_c;

  config_a = (Config *) malloc (sizeof(Config));
  config_b = (Config *) malloc (sizeof(Config));
  config_c = (Config *) malloc (sizeof(Config));

  while (answer == 'y' || answer == 'Y')
  {
  system("clear");
  introduction();
  system("clear");
    if (desired_case == 1)
    {
      scanf("%c",&response);
      printf("Do you desire to send output to a file (y/n)? ");
      scanf("%c",&response);
      if (response == 'y' || response == 'Y')
      {
        printf("\n\nEnter output file name: \n");
        scanf("%s",outdrxy);
        strcat(outdrxy,".xy");
        strcpy(outdrxz,outdrxy);
        strcat(outdrxz,".xz");
        printf("Opening %s for writing.\n",outdrxy);
        printf("Opening %s for writing.\n",outdrxz);
        drxyofp = fopen(outdrxy,"w");
        drxzofp = fopen(outdrxz,"w");
        send_to_file = 1;              /*global flag*/
        }
        case_1(config_a,config_b);
        if (response == 'y' || response == 'Y')
        {
          fclose(drxyofp);
          fclose(drxzofp);
        }
        response = 'n';
        send_to_file = 0;
       }
      if (desired_case == 2)
      {
        scanf("%c",&response);
        printf("Do you desire to send output to a file (y/n)? ");
        scanf("%c",&response);
        if (response == 'y' || response == 'Y')
        {
          printf("\n\nEnter output file name: \n");
          scanf("%s",outtemp);
          strcpy(outdrxy,outtemp);
          strcpy(outdrxz,outtemp);
          strcpy(outsim1,outtemp);
          strcpy(outsim2,outtemp);
          strcat(outdrxy,".xy");
          strcat(outdrxz,".xz");
          strcat(outsim1,".sim1");
          strcat(outsim2,".sim2");
          drxyofp = fopen(outdrxy,"w");
          drxzofp = fopen(outdrxz,"w");
          sim1ofp = fopen(outsim1,"w");
          sim2ofp = fopen(outsim2,"w");
          send_to_file = 1;
         }

        case_2(config_a,config_b);
        if (response == 'y' || response == 'Y')
        {
          fclose(drxyofp);
          fclose(drxzofp);
          fclose(sim1ofp);
          fclose(sim2ofp);
        }
        response = 'n';
        send_to_file = 0;
      }

    count = 0;  /*reinitialize array counter*/
    printf("\n\n\n\n\n");
    scanf("%c",&answer);  /*clear buffer*/
    printf("Do you want to enter more data (y or n)?  \n");
    scanf("%c",&answer);
    system("clear");
    }
  }

/****************************************************************************/
/*MAX_VAL..The following function determines the max of two values.       */
/****************************************************************************/
extern double max_val(value1,value2)
  double value1,value2;
{
  if (value1 > value2) return value1;
  if (value1 < value2) return value2;
  if (value1 == value2) return;
}

/****************************************************************************/
/*MIN_VAL..The following function determines the min of two values.       */
```

```
/*************************************************************************/
extern double min_val(value1,value2)
  double value1,value2;
{
  if (value1 < value2) return value1;
  if (value1 > value2) return value2;
  if (value1 == value2) return;
}

/*************************************************************************/
/*RAD_TO_DEG..The following function converts from radians to degrees.  */
/*************************************************************************/
extern double rad_to_deg(angle)
  double angle;
{
  return angle * 180.0 / PI;
}

/*************************************************************************/
/*DEG_TO_RAD..The following function converts from degrees to radians.  */
/*************************************************************************/
extern double deg_to_rad(angle)
  double angle;
{
  return angle * PI / 180.0;
}

/*************************************************************************/
/*NORMALIZE..The following function normalizes from -PI to PI           */
/*************************************************************************/
double normalize(angle)
  double angle;
{
  double x;

  x = angle;
  while (x  >  PI) x = x - PI - PI;
  while (x  < -PI) x = x + PI + PI;
  return x;
}
/*************************************************************************/
/*RELAZIMUTH..determines relative azimuth between two points            */
/*************************************************************************/
extern double relazimuth(pt1x,pt1y,pt2x,pt2y) /* range -PI .. PI */

   double pt1x,pt1y,pt2x,pt2y;  /* return normalized angle between points*/
{
   if ((pt1x == pt2x) && (pt1y == pt2y))
     return 0.0;
   else
     return normalize (atan2 (pt2y - pt1y, pt2x - pt1x));
}

/*************************************************************************/
/*RELELEV..determines relative elevation between two points in 3-D      */
/*************************************************************************/
extern double relelev(pt1x,pt1y,pt1z,pt2x,pt2y,pt2z)
                                             /* range -PI/2 .. PI/2 */
   double pt1x,pt1y,pt1z,pt2x,pt2y,pt2z;
                              /* return normalized angle between points*/
{
   if ((pt1x == pt2x) && (pt1y == pt2y) && (pt1z == pt2z))
     return 0.0;
   else
     return           (atan2 ((pt2z - pt1z),
                              (sqrt((pt2x - pt1x)*(pt2x - pt1x)+
                                    (pt2y - pt1y)*(pt2y - pt1y)))));
}
```

```
/****************************************************************************/
/*PRECEDE..determines angle precedence. Angles are in degrees.            */
/*Returns true if angle 1 preceeds angle 2                                */
/****************************************************************************/
extern int precede (angle1, angle2)
  double angle1, angle2;
{

  if (normalize (normalize (angle2) - normalize (angle1)) > 0.0)
    return TRUE;
  else
    return FALSE;
}

/****************************************************************************/
/*COMPUTE_NEW_CONFIGURATION..The following function computes the new       */
/*position and orientation of a robot given its starting point, speed      */
/*turn direction, turn rate pitch direction, pitch rate and desired DR     */ /
*interval.                                                                 */
/****************************************************************************/
extern Config compute_new_configuration(config_a,config_b,speed,
                                        turn_direction,turn_rate,
                                        pitch_direction,pitch_rate,
                                        delta_time)

  Config *config_a,*config_b;
  float  speed,turn_rate,pitch_rate,delta_time;
  int    turn_direction,pitch_direction;

{

 double delta_d,psi_v;

  delta_d = speed * delta_time;  /*distance traveled in interval*/
  if (turn_direction == -1)
    config_b->azimuth = normalize(config_a->azimuth +
                            (turn_rate * delta_time));
  if (turn_direction == 1)
    config_b->azimuth = normalize(config_a->azimuth -
                            (turn_rate * delta_time));
  if (turn_direction == 0)
    config_b->azimuth = config_a->azimuth;
  if (pitch_direction == -1)
    config_b->elevation = normalize(config_a->elevation +
                            (pitch_rate * delta_time));
  if (pitch_direction == 1)
    config_b->elevation = normalize(config_a->elevation -
                            (pitch_rate * delta_time));
  if (pitch_direction == 0)
    config_b->elevation = config_a->elevation;
  config_b->x = config_a->x + delta_d * cos(config_a->azimuth);
  config_b->y = config_a->y + delta_d * sin(config_a->azimuth);
  config_b->z = config_a->z + delta_d * sin(config_a->elevation);
  if (count == 0)
  {
    Dr[0].x = config_a->x;
    Dr[0].y = config_a->y;
    Dr[0].z = config_a->z;
    Dr[0].azimuth = config_a->azimuth;
    Dr[0].elevation = config_a->elevation;
    count = count + 1;
  }
  Dr[count].x = config_b->x;
  Dr[count].y = config_b->y;
  Dr[count].z = config_b->z;
  Dr[count].azimuth = config_b->azimuth;
  Dr[count].elevation = config_b->elevation;
  count = count + 1;
  return *config_b;
}

/***************************************************************************/
/*OUTPUT_XY_DR_POINTS..Sends dr points to output file                     */
/***************************************************************************/
extern void output_xy_dr_points(speed,turn_rate,delta_time)

  float speed,turn_rate,delta_time;
{
  int i;
  float max_x,
        max_y;

  if (send_to_file == 1)
  {
      fprintf(drxyofp,"%f ",Dr[0].x);  /*get Start printed in right place*/
      fprintf(drxyofp,"%f\n",Dr[0].y);
    for (i = 0; i <= count - 1; ++i)
    {
      if (i == 0)
      {
        fprintf(drxyofp,"\"Start_Track_S=%f_Tr=%f_Dt=%f\"\n",
                  speed,rad_to_deg(turn_rate),delta_time);
        fprintf(drxyofp,"%f",Dr[i].x);
        fprintf(drxyofp," %f\n",Dr[i].y);
      }
      else
      {
        fprintf(drxyofp,"%f",Dr[i].x);
        fprintf(drxyofp," %f\n",Dr[i].y);
      }
    }
    fprintf(drxyofp,"End_Track\n");
    if (desired_case == 2)
                            /*also need to enter waypoints*/
    {
      fprintf(drxyofp,"%f",Wpt[0].x);
      fprintf(drxyofp," %f\n",Wpt[0].y);
      fprintf(drxyofp,"First_Wpt\n");
      for (i = 0; i < total_wpts ; ++i)
      {
        fprintf(drxyofp,"%f",Wpt[i].x);
        fprintf(drxyofp," %f\n",Wpt[i].y);
      }
      fprintf(drxyofp,"Last_Wpt\n");
    }
    if (desired_case == 3)
                            /*also need to enter waypoints/obstacle*/
    {
      fprintf(drxyofp,"%f",Wpt[0].x);
      fprintf(drxyofp," %f\n",Wpt[0].y);
      fprintf(drxyofp,"First_Wpt\n");
      for (i = 0; i < total_wpts ; ++i)
      {
        fprintf(drxyofp,"%f",Wpt[i].x);
        fprintf(drxyofp," %f\n",Wpt[i].y);
      }
      fprintf(drxyofp,"Last_Wpt\n");
    }
  }
  return;
}

/***************************************************************************/
/*OUTPUT_XZ_DR_POINTS..Sends dr points to output file                     */
/***************************************************************************/
```

```
extern void output_xz_dr_points(speed,pitch_rate,delta_time)

  float speed,pitch_rate,delta_time;
{
  int i;
  float max_x,
        max_z;

  if (send_to_file == 1)
  {
    fprintf(drxzofp,"%f ",Dr[0].x);  /*get Start printed in right place*/
    fprintf(drxzofp,"%f\n",Dr[0].z);
    for (i = 0; i <= count - 1; ++i)
    {
      if (i == 0)
      {
        fprintf(drxzofp,"\"Start_Track_S=%f_Tr=%f_Dt=%f\"\n",
                speed,rad_to_deg(pitch_rate),delta_time);
        fprintf(drxzofp,"%f",Dr[i].x);
        fprintf(drxzofp," %f\n",Dr[i].z);
      }
      else
      {
        fprintf(drxzofp,"%f",Dr[i].x);
        fprintf(drxzofp," %f\n",Dr[i].z);
      }
    }
    fprintf(drxzofp,"End_Track\n");
    if (desired_case == 2)
                          /*also need to enter waypoints*/
    {
      fprintf(drxzofp,"%f",Wpt[0].x);
      fprintf(drxzofp," %f\n",Wpt[0].z);
      fprintf(drxzofp,"First_Wpt\n");
      for (i = 0; i < total_wpts ; ++i)
      {
        fprintf(drxzofp,"%f",Wpt[i].x);
        fprintf(drxzofp," %f\n",Wpt[i].z);
      }
      fprintf(drxzofp,"Last_Wpt\n");
    }
  }
  return;
}

/**************************************************************************/
/*OUTPUT_SIM1_POINTS..Sends points to simulator file                      */
/**************************************************************************/
extern void output_siml_points()

{

  int i;

    if (desired_case == 2)
    {
      for (i = 0; i <= count - 1 ; ++i)
      {
      fprintf(simlofp,"%f",(float)i/10.0);
      fprintf(simlofp," %f",Dr[i].x);
      fprintf(simlofp," %f",Dr[i].y);
      fprintf(simlofp," %f",Dr[i].z);
      fprintf(simlofp," %f",0.0);
      fprintf(simlofp," %f",Dr[i].elevation*(-1.0));
      fprintf(simlofp," %f",Dr[i].azimuth);
      fprintf(simlofp," %f %f %f %f %f %f %f %f %f %f\n",
              0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0);
      }
      }
return;
}

/****************************************************************************/
/*OUTPUT_ZTIME_POINTS..Sends z and time information to output file        */
/****************************************************************************/
extern void output_ztime_points()

{

  int i;

    if (desired_case == 2)
    {
      for (i = 0; i <= count - 1 ; ++i)
      {
      fprintf(sim2ofp," %f %f\n",(float)i/10.0,Dr[i].z);
      }
    }
return;
}

/****************************************************************************/
/*ENTER_WAYPOINTS..User enters desired waypoints.                           */
/****************************************************************************/
void enter_waypoints()

{
  Point  point1,point2,point3;
  int i;
  int wpt_number = 0;
  float a,b,c;          /*floats for scanf to be cast to doubles*/

  printf("\n\n\n\n\n");
  printf("Enter the desired number of waypoints (ten maximum): ");
  scanf("%d",&total_wpts);
  system("clear");
  printf("Please enter the x, y and z coordinates of you waypoints\n");
  printf("in sequential order.\n\n");
  for (i = 0; i < total_wpts; ++i)
  {
    (wpt_number = i+1);
    printf("\n\nWaypoint number %d: \n",wpt_number);
    printf("  x = ");
    scanf("%f",&a);
    printf("\n  y = ");
    scanf("%f",&b);
    printf("\n  z = ");
    scanf("%f",&c);
    point1.x = (double) a;
    point1.y = (double) b;
    point1.z = (double) c;
    Wpt[i].x = point1.x;
    Wpt[i].y = point1.y;
    Wpt[i].z = point1.z;
    if (i > 0 && i < total_wpts)
    {
      point2.x = Wpt[i - 1].x;
      point2.y = Wpt[i - 1].y;
      point2.z = Wpt[i - 1].z;
      point3.x = Wpt[i].x;
      point3.y = Wpt[i].y;
      point3.z = Wpt[i].z;
      point1.azimuth = relazimuth(point2.x,point2.y,
                                  point3.x,point3.y);
      Wpt[i-1].azimuth = point1.azimuth;
    }
  }
  Wpt[total_wpts - 1].azimuth = Wpt[total_wpts - 2].azimuth;
  /*sets the last waypoint's relazimuth to that of the next to last*/
  printf("Your waypoints are as follows:\n\n");
  for (i = 0; i < total_wpts; ++i)
  {
    printf("Waypoint %d = (%f,%f,%f,%f)\n",
                             (i + 1),Wpt[i].x,Wpt[i].y,Wpt[i].z,
                             rad_to_deg(Wpt[i].azimuth));
  }
  printf("\n\n\n\n\n");
  return;
}

/**************************************************************************/
/*RETRIEVE_WAYPOINTS..Retrieves waypoint data from file                  */
/**************************************************************************/
extern void retrieve_waypoints()

{

  Point  point1,point2,point3;
  int i;
  int wpt_number = 0;
  float a,b,c;          /*floats for scanf to be cast to doubles*/

  wptifp = fopen(inwpt,"r");
  while(fscanf(wptifp, "%f%f%f", &a,&b,&c) != EOF)
  {
    point1.x = (double) a;
    point1.y = (double) b;
    point1.z = (double) c;
    Wpt[wpt_number].x = point1.x;
    Wpt[wpt_number].y = point1.y;
    Wpt[wpt_number].z = point1.z;
    if (wpt_number > 0)
    {
      point2.x = Wpt[wpt_number - 1].x;
      point2.y = Wpt[wpt_number - 1].y;
      point2.z = Wpt[wpt_number - 1].z;
      point3.x = Wpt[wpt_number].x;
      point3.y = Wpt[wpt_number].y;
      point3.z = Wpt[wpt_number].z;
      point1.azimuth = relazimuth(point2.x,point2.y,
                                  point3.x,point3.y);
      Wpt[wpt_number-1].azimuth = point1.azimuth;
    }
  wpt_number = wpt_number + 1;
  }
  Wpt[wpt_number -1 ].azimuth = Wpt[wpt_number - 2].azimuth;
  /*sets the last waypoint's relative azimuth to that of the next to last*/
  printf("Your waypoints are as follows:\n\n");
  for (i = 0; i < wpt_number; ++i)
  {
    printf("Waypoint %d = (%f,%f,%f,%f)\n",
                             (i + 1),Wpt[i].x,Wpt[i].y,Wpt[i].z,
                             rad_to_deg(Wpt[i].azimuth));
  }
  printf("\n\n\n\n\n");
  fclose(wptifp);
  count = wpt_number;
  total_wpts = wpt_number;
  return;
}

/**************************************************************************/
/*DETERMINE_NEXT_WAYPOINT..determines next waypoint. If past             */
/*last waypoint in list, next waypoint is first waypoint.                */
```

```
/*****************************************************************************/
extern int determine_next_waypoint(config_a)

  Config *config_a;

{

  Point point1,point2;
  int next_waypoint,i;
  int direction = 0;        /*1=east,-1=west*/
  float robot_to_point_relazimuth,checking_relazimuth_one,
        checking_relazimuth_two;

  /*increment through wpts until checked wpt is behind robot*/
  /*the next waypoint is then the next desired wpt*/
  for (i = last_visited_wpt + 1 ; i < total_wpts; i = last_visited_wpt + 1)
  {
    point1.x = config_a->x;
    point1.y = config_a->y;
    point1.z = config_a->z;
    if (verbose == TRUE)
    {
      printf("Your input point is (%f,%f,%f,%f)\n",
                               point1.x,point1.y,point1.z,
                               rad_to_deg(config_a->azimuth));
    }
    point2.x = Wpt[i].x;
    point2.y = Wpt[i].y;
    point2.z = Wpt[i].z;
    if (verbose == TRUE)
    {
      printf("Analyzed waypoint is (%f,%f,%f,%f)\n",
                               Wpt[i].x,Wpt[i].y,Wpt[i].z,
                               rad_to_deg(Wpt[i].azimuth));
    }
    point2.azimuth = Wpt[i].azimuth;
    robot_to_point_relazimuth = relazimuth(point1.x,point1.y,
                                      point2.x,point2.y);
    /*check to see if robot is past waypoint*/
    if (i == 0) /*use relazimuth of wpt 0 as reference*/
    {
      if (rad_to_deg(point2.azimuth) <= 90 &&
          rad_to_deg(point2.azimuth) >= -90) /*heading east*/
      {
        checking_relazimuth_one = normalize(point2.azimuth - PI/2);
        checking_relazimuth_two = normalize(point2.azimuth + PI/2);
        direction = 1;
      }
      else                                  /*heading west*/
      {
        checking_relazimuth_one = normalize(point2.azimuth + PI/2);
        checking_relazimuth_two = normalize(point2.azimuth - PI/2);
        direction = -1;
      }
    }
    if (i != 0) /*use previous wpt relazimuth as reference*/
    {
      if (rad_to_deg(Wpt[i-1].azimuth) <= 90 &&
          rad_to_deg(Wpt[i-1].azimuth) >= -90) /*heading east*/
      {
        checking_relazimuth_one = normalize(Wpt[i-1].azimuth - PI/2);
        checking_relazimuth_two = normalize(Wpt[i-1].azimuth + PI/2);
        direction = 1;
      }
      else /*heading west*/
      {
        checking_relazimuth_one = normalize(Wpt[i-1].azimuth + PI/2);
        checking_relazimuth_two = normalize(Wpt[i-1].azimuth - PI/2);
      direction = -1;
    }
  }
  if (verbose == TRUE)
  {
    printf("Robot to point relazimuth is %f \n",
             rad_to_deg(robot_to_point_relazimuth));
    printf("Checking relazimuth one is %f\n",
             rad_to_deg(checking_relazimuth_one));
    printf("Checking relazimuth two is %f\n",
             rad_to_deg(checking_relazimuth_two));
  }
  if (direction == 1) /*heading east*/
  {
    if ((robot_to_point_relazimuth <= checking_relazimuth_one) ||
        (robot_to_point_relazimuth >= checking_relazimuth_two))
       /*robot is down track from waypoint being analyzed*/
    {
      last_visited_wpt = i;
      if (verbose == TRUE)
        printf("You are ahead of analyzed point\n");
        if (i == total_wpts - 1)
      {
        if (verbose == TRUE)
          printf("You are beyond last waypoint\n");
        next_waypoint = -1;
        return next_waypoint;
      }                             /*robot is past last waypoint */
    }                                             /*terminate track*/
     if ((robot_to_point_relazimuth > checking_relazimuth_one) &&
       (robot_to_point_relazimuth < checking_relazimuth_two))
     {
       if (verbose == TRUE)
         printf("You are behind analyzed point\n");
       next_waypoint = i;
       if (verbose == TRUE)
         printf("Check waypoint is %d\n",next_waypoint);
         return next_waypoint;
      }
  }
  if (direction == -1) /*heading west*/
  {
    if ((robot_to_point_relazimuth <= checking_relazimuth_two) &&
        (robot_to_point_relazimuth >= checking_relazimuth_one))
       /*robot is down track from waypoint being analyzed*/
    {
     last_visited_wpt = i;
      if (verbose == TRUE)
        printf("You are ahead of analyzed point\n");
      if (i == total_wpts - 1)
      {
        if (verbose == TRUE)
          printf("You are beyond last waypoint\n");
        next_waypoint = -1;
        return next_waypoint;
      }                              /*robot is past last waypoint */
    }                                              /*terminate track*/
    if ((robot_to_point_relazimuth > checking_relazimuth_two) ||
       (robot_to_point_relazimuth < checking_relazimuth_one))
    {
      if (verbose == TRUE)
        printf("You are behind analyzed point\n");
      next_waypoint = i;
      if (verbose == TRUE)
        printf("Check waypoint is %d\n",next_waypoint);
         return next_waypoint;
    }
  }
  }
}

/**************************************************************************/
/*PROCEED_ALONG_PATH..Determines which waypoint is next then steers to   */
/*that waypoint. Repeats until all remaining waypoints have been reached.*/
/**************************************************************************/
extern void proceed_along_path(config_a,config_b,turn_rate,pitch_rate,
                               speed,delta_time)

  Config *config_a,*config_b;
  float turn_rate,pitch_rate,speed,delta_time;
{

  int next_waypoint,next_wpt,i;
  Point point1,point2;
  float alpha,          /*relazimuth angle between robot and desired wpt*/
        beta,           /*pitch angle between robot and desired wpt*/
        theta,          /*elevation of robot (pitch)*/
        psi;            /*azimuth of robot*/
  int   turn_direction, /*1=right,-1=left,0=none*/
        pitch_direction; /*-1=down,1=up,0=none*/

  i = determine_next_waypoint(config_a);
                        /*determine first wpt to steer to*/
  if (i == -1) return;
  next_wpt = (i + 1);
  if (verbose == TRUE)
    printf("The next waypoint number is:  %d\n\n",next_wpt);
  while (i <  - 1 || i >  - 1)
                                  /*proceed from next til last wpt*/
  {
    i = determine_next_waypoint(config_a);
    if (i == -1)  /*flag indicating through with track*/
      return;
    next_wpt = (i + 1);
    if (verbose == TRUE)
      printf("The next waypoint number is:  %d\n\n",next_wpt);
    point1.x = config_a->x;
    point1.y = config_a->y;
    point1.z = config_a->z;
    point2.x = Wpt[i].x;
    point2.y = Wpt[i].y;
    point2.z = Wpt[i].z;
    alpha = relazimuth(point1.x,point1.y,point2.x,point2.y);
    beta = relelev(point1.x,point1.y,point1.z,
                        point2.x,point2.y,point2.z);
    psi = config_a->azimuth;
    theta = config_a->elevation;
    if (precede(alpha,psi) == 1)
      {turn_direction = 1;           /*turn right*/
       if (verbose == TRUE)
         printf("Must turn right!\n");}
    if (precede(psi,alpha) == 1)
      {turn_direction = -1;          /*turn left*/
       if (verbose == TRUE)
         printf("Must turn left!\n");}
    if (normalize(psi) == normalize(alpha + PI))
      {turn_direction = 0;
     if (verbose == TRUE)
       printf("Lets turn left!\n");}
    if (precede(beta,theta) == 1)
      {pitch_direction = -1;          /*turn down*/
       if (verbose == TRUE)
         printf("Must turn down!\n");}
    if (precede(theta,beta) == 1)
      {pitch_direction = 1;          /*turn up*/
     if (verbose == TRUE)
```

```
        printf("Must turn up!\n");}
     if (normalize(theta) == normalize(beta + PI))
       {pitch_direction = 0;
      if (verbose == TRUE)
        printf("Lets turn down!\n");}
     compute_new_configuration(config_a,config_b,speed,
                              turn_direction,turn_rate,
                              (pitch_direction*(-1)), pitch_rate,delta_time);
     config_a->x = config_b->x;
     config_a->y = config_b->y;
     config_a->z = config_b->z;
     config_a->azimuth = config_b->azimuth;
     config_a->elevation = config_b->elevation;
   }
}

/***************************************************************************/
/*INTRODUCTION..Lead in to guidance program                                */
/***************************************************************************/
extern int introduction()
{

 printf("============================================================\n");
 printf("============================================================\n\n");
 printf("Title:  Guidance\n");
 printf("Author: Mark Compton\n");
 printf("Course: Thesis\n");
 printf("Date:   09 March 92\n\n");
 printf("Description:\n\n");
 printf("This program analyzes the path of a point robot as it maneuvers\n");
 printf("from waypoint to waypoint. It then outputs this information to \n");
 printf("a graph file where it can be printed.\n\n");
 printf("The objectives are as follows:\n\n");
 printf("Case 1: Determine path of robot based on initial configuration,\n");
 printf("        speed, turn direction, turn rate and travel time. \n\n");
 printf("Case 2: Determine path of robot direct from start point to first\n");
 printf("        available waypoint, then from waypoint to waypoint.\n\n");
 printf("============================================================\n");
 printf("============================================================\n\n");
 printf("Please enter the number of the case you wish to analyze: ");
 scanf("%d",&desired_case);
}
/***************************************************************************/
/*CASE_1..Function to i/o and execute case 1                               */
/***************************************************************************/
extern void case_1(config_a,config_b)

  Config *config_a,*config_b;

{

  int    i,
         turn_direction, /*-1=left,0=none,1=right*/
         pitch_direction;/*-1=down,0=none,1=up*/
  float  time,           /*total time of analysis*/
         speed,          /*desired robot speed*/
         turn_rate,      /*degrees per second*/
         pitch_rate,     /*degrees per second*/
         delta_time;     /*time interval between DR positions*/
  float  posx,posy,posz,orient,elev,
         d,e,f,g;           /*floats for scanf to be cast to doubles*/

  printf("\n\n\n\n");
  printf("                                  ========\n");
  printf("                                  =CASE 1=\n");
  printf("                                  ========\n\n\n\n\n");
  printf("This case analyzes the robot path given a start position\n");
  printf("(x,y,z), start azimuth with respect to the x-axis(+-180), \n");
printf("(x,y,z), start elevation with respect to the x-axis(+-180), \n");
printf("turn rate (deg per sec),and travel time (seconds).\n\n\n\n\n");
printf("Enter the starting x, y and z coordinates of your robot.\n\n ");
printf("start x = ");
scanf("%f",&posx);
printf("\nstart y = ");
scanf("%f",&posy);
printf("\nstart z = ");
scanf("%f",&posz);
system("clear");
printf("\n\n\n\nEnter the start azimuth (+-180) of your robot.\n\n");
printf("start azimuth = ");
scanf("%f",&orient);
printf("\n\n\n\nEnter the start elevation (+-180) of your robot.\n\n");
printf("start elevation = ");
scanf("%f",&elev);
config_a->x = (double) posx;
config_a->y = (double) posy;
config_a->z = (double) posz;
config_a->azimuth = deg_to_rad(orient);
config_a->elevation = deg_to_rad(elev);
system("clear");
printf("\n\n\n\nEnter in the speed of your robot (units per second).\n\n");
printf("speed = ");
scanf("%f",&speed);
system("clear");
printf("\n\n\n\nEnter in the turn direction of your robot.\n");
printf("(-1 = left, 1 = right, 0 = no turn).\n\n");
printf("turn direction = ");
scanf("%d",&turn_direction);
system("clear");
printf("\n\n\n\n");
printf("Enter in the turn rate of your robot (degrees per second).\n");
printf("Note: use turn rate of 0.0 if no turn.\n\n");
printf("turn rate = ");
scanf("%f",&f);
turn_rate = deg_to_rad(f);
system("clear");
printf("\n\n\n\nEnter in the pitch direction of your robot.\n");
printf("(-1 = down, 1 = up, 0 = no turn).\n\n");
printf("pitch direction = ");
scanf("%d",&pitch_direction);
system("clear");
printf("Enter in the pitch rate of your robot (degrees per second).\n");
printf("Note: use pitch rate of 0.0 if no change in pitch.\n\n");
printf("pitch rate = ");
scanf("%f",&g);
pitch_rate = deg_to_rad(g);
system("clear");
printf("\n\n\n\n");
printf("Enter the time of travel for your robot (seconds). \n\n");
printf("time = ");
scanf("%f",&time);
system("clear");
printf("\n\n\n\n");
printf("Enter the time interval between DR positions (seconds).\n\n");
printf("interval = ");
scanf("%f",&delta_time);
system("clear");
printf("\n\nYour data entries are as follows: \n\n");
printf("Start coordinates = (%f,%f,%f)\n",
                        config_a->x,config_a->y,config_a->z);
printf("Start azimuth = %f\n",rad_to_deg(config_a->azimuth));
printf("Start elevation = %f\n",rad_to_deg(config_a->elevation));
printf("Speed = %f\n",speed);
printf("Turn direction = %d\n",turn_direction);
printf("Turn rate = %f\n",rad_to_deg(turn_rate));
printf("Pitch direction = %d\n",pitch_direction);
```

```
   printf("Pitch rate = %f\n",rad_to_deg(pitch_rate));
   printf("Travel time = %f\n", time);
   printf("Time interval for DR = %f\n\n",delta_time);
    for (i = 1; i <= time / delta_time; ++i)
    {
     compute_new_configuration(config_a,config_b,speed,
                               turn_direction,turn_rate,
                               (pitch_direction*(-1)),pitch_rate,delta_time);

    config_a->x = config_b->x;
    config_a->y = config_b->y;
    config_a->z = config_b->z;
    config_a->azimuth = config_b->azimuth;
    config_a->elevation = config_b->elevation;
    }
   printf("\n\n\n\n\n\n\n\n\n");
   output_xy_dr_points(speed,turn_rate,delta_time);
   output_xz_dr_points(speed,pitch_rate,delta_time);
   count = 0;
   printf("The robot's end configuration at time +%f is (%f,%f,%f,%f,%f).\n",
           time,config_b->x,config_b->y,config_b->z,
           rad_to_deg(config_b->azimuth),
           rad_to_deg(config_b->elevation));
   return;
}

/************************************************************************/
/*CASE_2..Function to i/o and execute case 2                            */
/************************************************************************/
extern void case_2(config_a,config_b)

   Config *config_a,*config_b;
{

   float   speed,          /*desired robot speed*/
           turn_rate,      /*degrees per second*/
           pitch_rate,     /*degrees per second*/
           delta_time;     /*time interval between DR positions*/
   float   posx,posy,posz,orient,elev,
           d,e,f,g;    /*floats for scanf to be cast to doubles*/
   int     next_wpt;
   char    ans = 'y',
           reply = 'n';

   printf("\n\n\n\n");
   printf("                                     ========\n");
   printf("                                     =CASE 2=\n");
   printf("                                     ========\n\n\n\n\n");
   printf("This case analyzes the robot path given a start position\n");
   printf("(x,y,z), start azimuth with respect to the x-axis(+-180), \n");
   printf("(x,y,z), start elevation with respect to the x-axis(+-180), \n");
   printf("turn rate (deg per sec), and desired waypoints.\n\n\n\n");
   printf("The robot determines which waypoint is next at a given\n");
   printf("time, proceeds directly to that waypoint and then proceeds\n");
   printf("along the planned route until the final waypoint is achieved.\n");
   printf("\n\n\n\n\n");
   printf("Enter the starting x, y and z coordinates of your robot.\n\n ");
   printf("start x = ");
   scanf("%f",&posx);
   printf("\nstart y = ");
   scanf("%f",&posy);
   printf("\nstart z = ");
   scanf("%f",&posz);
   system("clear");
   printf("\n\n\n\nEnter the start azimuth (+-180) of your robot.\n\n");
   printf("start azimuth = ");
   scanf("%f",&orient);
   printf("\n\n\n\nEnter the start elevation (+-180) of your robot.\n\n");
```

```
    printf("start elevation = ");
    scanf("%f",&elev);
    config_a->x = (double) posx;
    config_a->y = (double) posy;
    config_a->z = (double) posz;
    config_a->azimuth = deg_to_rad(orient);
    config_a->elevation = deg_to_rad(elev);
    system("clear");
    printf("\n\n\n\nEnter in the speed of your robot (units per second).\n\n");
    printf("speed = ");
    scanf("%f",&speed);
    system("clear");
    printf("\n\n\n\n");
    printf("Enter in the turn rate of your robot (degrees per second).\n\n");
    printf("turn rate = ");
    scanf("%f",&f);
    turn_rate = deg_to_rad(f);
    system("clear");
    printf("Enter in the pitch rate of your robot (degrees per second).\n\n");
    printf("pitch rate = ");
    scanf("%f",&g);
    pitch_rate = deg_to_rad(g);
    system("clear");
    printf("\n\n\n\n");
    printf("Enter the time interval between DR positions (seconds).\n\n");
    printf("interval = ");
    scanf("%f",&delta_time);
    system("clear");
    printf("\n\nYour data entries are as follows: \n\n");
    printf("Start coordinates = (%f,%f,%f)\n",
                             config_a->x,config_a->y,config_a->z);
    printf("Start azimuth = %f\n",rad_to_deg(config_a->azimuth));
    printf("Start elevation = %f\n",rad_to_deg(config_a->elevation));
    printf("Speed = %f\n",speed);
    printf("Turn rate = %f\n",rad_to_deg(turn_rate));
    printf("Pitch rate = %f\n",rad_to_deg(pitch_rate));
    printf("Time interval for DR = %f\n\n",delta_time);
    printf("\n\n\n\n\n");
    scanf("%c",&reply);
    printf("Will you be retrieving waypoints from a file? ");
    scanf("%c",&reply);
    if (reply == 'y' || reply == 'Y')
    {
      printf("Enter the waypoint input file name: ");
      scanf("%s",inwpt);
      retrieve_waypoints();
    }
    else
      enter_waypoints();
    scanf("%c",&ans);
    printf("\n\nWhen ready to proceed with analysis enter 'y': ");
    scanf("%c",&ans);
    system("clear");
    proceed_along_path(config_a,config_b,turn_rate,
                       pitch_rate,speed,delta_time);
    printf("\nThe robot has reached the final waypoint.\n");
    output_xy_dr_points(speed,turn_rate,delta_time);
    output_xz_dr_points(speed,pitch_rate,delta_time);
    output_siml_points();
    output_ztime_points();
    count = 0;
    return;
  }
```

APPENDIX F: AUTONOMOUS SONAR CLASSIFICATION EXPERT SYSTEM SOURCE CODE

```
;_____________________________________________________________________
;
;                  AUV Sonar Expert System
;
;   Filename:      auvsonar
;
;   Purpose:       Batch file for auvsonar.clp which resets and executes the
;                  AUV sonar contact classification expert system.
;
;   Paper:         "Autonomous Underwater Vehicle Sonar Classification using
;                   Expert Systems and Neural Networks"
;
;                  IEEE OCEANS '92 Conference, Newport, Rhode Island
;
;   Authors:       Don Brutzman, Mark Compton and Dr. Yutaka Kanayama
;
;   Date:          24 November 91
;
;   Execution:     unix>  clips5                    unix>  clips5
;                  CLIPS> (load auvsonar.clp)       CLIPS> (batch auvsonar)
;                  CLIPS> (reset)                   CLIPS> (run)
;                  CLIPS> (run)
;
;_____________________________________________________________________

; Clear & close files in case they were left open during previous execution

        (clear)              ; clear all facts and rules
(close rangefile)     ; Close AUV-recorded pool test data input file
(close plotfile)      ; Close xy coordinate file used for graph output
(close auvfile)       ; Close expert system classification output file

;_____________________________________________________________________

        (load auvsonar.clp)  ; Load in AUV Sonar Classification Expert System

        (undefrule oldarea1)
        (undefrule oldarea2)

        (reset)              ; Initialize agenda and assert initial facts

;;;;;   (run)                ; Execute AUV Sonar Classification Expert System
```

```
;-----------------------------------------------------------------------------
;
;                AUV Sonar Expert System
;
;  Filename:     auvsonar.clp
;
;  Purpose:      Define data templates, rules, functions and user interface
;                for the AUV sonar contact classification expert system.
;
;  Paper:        "Autonomous Sonar Classification using Expert Systems"
;                IEEE Oceanic Engineering Society
;                IEEE OCEANS '92 Conference, Newport, Rhode Island
;
;  Authors:      Don Brutzman and Mark Compton
;  Advisor:      Dr. Yutaka Kanayama
;
;  Date:         1 March 91
;
;  Comments:     This expert system takes data files generated by the NPS AUV,
;                uses sonar returns and AUV position to generate locations of
;                sonar contacts, perform two-dimensional linear regression to
;
;                build line segments, combine segments into polyhedrons and
;                then determines the probable classification of each polyhedron.
;
;  Language:     CLIPS "C" Language Integrated Production System
;
;  Execution:     unix>  clips5                  |    unix>  clips5
;                CLIPS> (load auvsonar.clp)      |   CLIPS> (batch auvsonar)
;                CLIPS> (reset)                  |   CLIPS> (run)
;                CLIPS> (run)
;
;                Execution 'dribble' files are saved in auvsonar.log
;
;  References:   _Sonar Data Interpretation for Autonomous Mobile Robots_,
;                Yutaka Kanayama, Tetsuo Noguchi, and Bruce Hartman,
;                unpublished paper.
;
;  History:      Original program development for CS4311 Expert Systems
;                taught by Dr. Kanayama.
;
;  Caveat:       The NPS pool coordinate system is the world reference used
;                where x is pool length, y is pool width, and z is pool depth.
;
;  Status:       Initial development complete for object classification.
;                Full pool depth used for pool object outputs.
;                Initial offset option for centering pool data included.
;                Verbose output option and excess data retraction completed.
;                Gyro error/gyro drift rate evaluation & correction implemented.
;                Centroid and cross-sectional area calculations done for objects.
;                Top-level classification of objects using area is possible.
;                Mine classification implemented satisfactorily.
;                Excessively narrow objects are reclassified as walls.
;
;-----------------------------------------------------------------------------


;-----------------------------------------------------------------------------
;
;    Data Type Deftemplates
;
;-----------------------------------------------------------------------------
;
;    Data template and slot names correspond to AUV Data Dictionary definitions.
;    Data template names have their first letter capitalized.
;    Variable names are all lower case.
;    CLIPS data types and symbols used in symbolic slots are capitalized.
;
```

```
;___________________________________________________________________________

(deftemplate Range_data

  (field time                                 ; time is positive, set by AUV
                  (type    NUMBER)
                  (default 0)                 ; time zero is used for dummy facts
                  (range   0 ?VARIABLE))
  (field x                                    ; element of Point_3D AUV data type
                  (type    NUMBER)            ; dead reckoning estimate of travel
                  (default 0)                 ;    relative to start position
                  (range   0 ?VARIABLE))
  (field y                                    ; element of Point_3D AUV data type
                  (type    NUMBER)            ; dead reckoning estimate of travel
                  (default 0)                 ;    relative to start position
                  (range   0 ?VARIABLE))
  (field z                                    ; element of Point_3D AUV data type
                  (type    NUMBER)            ; source:  pressure-sensing depth cell
                  (default 0)                 ; which may be inaccurate when shallow
                  (range   0 ?VARIABLE))
  (field phi                                 ; element of Attitude_3D AUV data type
                  (type    NUMBER)            ;   in radians
                  (default 0))                ;     (roll)
  (field theta                               ; element of Attitude_3D AUV data type
                  (type    NUMBER)            ;   in radians
                  (default 0))                ;     (pitch)
  (field psi                                 ; element of Attitude_3D AUV data type
                  (type    NUMBER)            ;   in radians.  Note caveat on pg. 1
                  (default 0))                ;     (yaw)
  (field p                                    ; element of Point_3D AUV data type
                  (type    NUMBER)            ;   in radians/sec
                  (default 0)
                  (range   0 ?VARIABLE))
  (field q                                    ; element of Point_3D AUV data type
                  (type    NUMBER)            ;   in radians/sec
                  (default 0)
                  (range   0 ?VARIABLE))
  (field r                                    ; element of Point_3D AUV data type
                  (type    NUMBER)            ;   in radians/sec
                  (default 0)
                  (range   0 ?VARIABLE))
  (field delta_dive_planes                    ; change in bow/stern planes position
                  (type    NUMBER)            ;   in degrees
                  (default 0)
                  (range   0 ?VARIABLE))
  (field delta_rudders                        ; change in rudder planes position
                  (type    NUMBER)            ;   in degrees
                  (default 0)
                  (range   0 ?VARIABLE))
  (field range_a                              ; 0-4095 range units correspond to
                  (type    NUMBER)            ;   0..30m pool or 0..300m ocean.
                  (default 0)
                  (range   0 ?VARIABLE))
  (field range_b                              ; Up to 4 transducers can be included.
                  (type    NUMBER)
                  (default 0)
                  (range   0 ?VARIABLE))
  (field range_c
                  (type    NUMBER)
                  (default 0)
                  (range   0 ?VARIABLE))
  (field range_d
                  (type    NUMBER)
                  (default 0)
                  (range   0 ?VARIABLE))
  (field valid_a                              ; Validity signal from sonar hardware
                  (type    INTEGER)
                  (default 1))
  (field valid_b
                    (type    INTEGER)
                    (default 1))
  (field valid_c
                    (type    INTEGER)
                    (default 1))
  (field valid_d
                    (type    INTEGER)
                    (default 1))
  (field speed                          ; AUV speed from flow sensor
                    (type    NUMBER)
                    (default 0.0))
  (field processed                      ; set TRUE when point is asserted,
                    (type    SYMBOL)    ;     FALSE until then.
                    (default FALSE)
          (allowed-values TRUE FALSE))
)

;________________________________________________________________________________

(deftemplate Object_data

  (field detection_time                  ; time is positive, set by AUV
                    (type    NUMBER)
                    (default 0)
                    (range   0 ?VARIABLE))
  (field latest_time
                    (type    NUMBER)
                    (default 0)
                    (range   0 ?VARIABLE))
  (field valid
                    (type    INTEGER)
                    (default 0))
  (field x                               ; object center
                    (type    NUMBER)
                    (default 0)
                    (range   0 ?VARIABLE))
  (field y                               ; object center
                    (type    NUMBER)
                    (default 0)
                    (range   0 ?VARIABLE))
  (field z                               ; object center
                    (type    NUMBER)
                    (default 0)
                    (range   0 ?VARIABLE))
  (field accuracy
                    (type    FLOAT)
                    (default 0.0)
                    (range   0.0 ?VARIABLE))
  (field object
                    (type    INTEGER)
                    (default 0)
                    (range   0 9))
  (field length
                    (type    NUMBER)
                    (default 0)
                    (range   0 ?VARIABLE))
  (field height
                    (type    NUMBER)
                    (default 0)
                    (range   0 ?VARIABLE))
  (field width
                    (type    NUMBER)
                    (default 0)
                    (range   0 ?VARIABLE))
  (field confidence                      ; normalized
                    (type    FLOAT)
                    (default 0.0)
```

```
                 (range   0.0 1.0))
)

;_______________________________________________________________________________

(deftemplate Point

  (field time                              ; time is positive, set by AUV
                 (type    NUMBER)
                 (default 0)
                 (range   0 ?VARIABLE))
  (field x                                 ; element of Point_3D AUV data type
                 (type    NUMBER)
                 (default 0)
                 (range   0 ?VARIABLE))
  (field y                                 ; element of Point_3D AUV data type
                 (type    NUMBER)
                 (default 0)
                 (range   0 ?VARIABLE))
  (field z                                 ; element of Point_3D AUV data type
                 (type    NUMBER)
                 (default 0)
                 (range   0 ?VARIABLE))
  (field valid
                 (type    INTEGER)
                 (default 0))
  (field status
                 (type    SYMBOL)
                 (default NEW)
          (allowed-values NEW ACTIVE INVALID ENDPOINT USED))
)

;_______________________________________________________________________________

(deftemplate Regression_line

  (field start                                  ; matches time of start point
                 (type    NUMBER)
                 (default 0)
                 (range   0 ?VARIABLE))
  (field end                                    ; matches time of end point
                 (type    NUMBER)
                 (default 0)
                 (range   0 ?VARIABLE))
  (field r
                 (type    FLOAT)
                 (default 0.0)
                 (range   0.0 ?VARIABLE))
  (field orientation                             ; normalized degrees
                 (type    FLOAT)
                 (default 0.0)
                 (range   0.0 ?VARIABLE))
  (field correlation
                 (type    FLOAT)
                 (default 0.0)
                 (range   0.0 ?VARIABLE))
  (field status
                 (type    SYMBOL)
                 (default NEW)
          (allowed-values NEW CURRENT VALID USED USED_FOR_AREA))
)

;_______________________________________________________________________________

(deftemplate Node

  (field time
                 (type    NUMBER)
                 (default 0)
                 (range   0 ?VARIABLE))
  (field x
                 (type    NUMBER)
                 (default 0)
                 (range   0 ?VARIABLE))
  (field y
                 (type    NUMBER)
                 (default 0)
                 (range   0 ?VARIABLE))
  (field z
                 (type    NUMBER)
                 (default 0)
                 (range   0 ?VARIABLE))
  (field accuracy
                 (type    FLOAT)
                 (default 0.0)
                 (range   0.0 ?VARIABLE))
  (field confidence
                 (type    FLOAT)
                 (default 0.0)
                 (range   0.0 1.0))
)

;________________________________________________________________________________

(deftemplate Edge

  (field start                      ; slot values are times corresponding to data
                 (type    FLOAT))
  (field end
                 (type    FLOAT))
  (field averagez
                 (type    NUMBER)
                 (default 0)
                 (range   0 ?VARIABLE))
  (field status
                 (type    SYMBOL)
                 (default USED)
          (allowed-values USED USED_FOR_AREA))
)

;________________________________________________________________________________

(deftemplate Curve                         ; not yet implemented

  (field time
                 (type    NUMBER)
                 (default 0)
                 (range   0 ?VARIABLE))
  (field node                      ; slot values are times corresponding to data
                 (type    FLOAT))
  (field edge
                 (type    FLOAT))
  (field shape
                 (type    SYMBOL))
)

;________________________________________________________________________________

(deftemplate Polyhedron

  (field start                      ; time of the initial node/edge/curve element
                 (type    NUMBER)
                 (default 0)
                 (range   0 ?VARIABLE))
  (field end                        ; time of most recent node/edge/curve element
                 (type    NUMBER)
                 (default 0)
                 (range   0 ?VARIABLE))
(field startx
                 (type    NUMBER)
                 (default 0)
                 (range   0 ?VARIABLE))
(field starty
                 (type    NUMBER)
                 (default 0)
                 (range   0 ?VARIABLE))
(field startz
                 (type    NUMBER)
                 (default 0)
                 (range   0 ?VARIABLE))
(field centroidx
                 (type    NUMBER)
                 (default 0)
                 (range   0 ?VARIABLE))
(field centroidy
                 (type    NUMBER)
                 (default 0)
                 (range   0 ?VARIABLE))
(field centroidz
                 (type    NUMBER)
                 (default 0)
                 (range   0 ?VARIABLE))
(field sidecount
                 (type    INTEGER)
                 (default 1)
                 (range   1 ?VARIABLE))
(field sidecounter1
                 (type    INTEGER)
                 (default -1)
                 (range   -1 ?VARIABLE))
(field sidecounter2
                 (type    INTEGER)
                 (default -1)
                 (range   -1 ?VARIABLE))
(field area
                 (type    FLOAT)
                 (default 0.0)
                 (range   0.0 ?VARIABLE))
(field height
                 (type    FLOAT)
                 (default 0.0)
                 (range   0.0 ?VARIABLE))
(field accuracy
                 (type    FLOAT)
                 (default 0.0)
                 (range   0.0 ?VARIABLE))
(field confidence
                 (type    FLOAT)
                 (default 0.0)
                 (range   0.0 1.0))
(field trait
                 (type    SYMBOL))
(field status
                 (type    SYMBOL)
                 (default ACTIVE)
          (allowed-values ACTIVE COMPLETE USED_FOR_AREA))
(field classification
                 (type    SYMBOL)
                 (default WALL)
          (allowed-values NEW CURRENT WALL OBJECT MINE SWIMMER UNKNOWN
                          SEA-MOUNT SUBMARINE SHIP_I_E_SKIMMER_PUKE
                          BIOLOGICS LOTS_OF_BIOLOGICS))
)
```

```
;________________________________________________________________________________
;
; Initialization of Flag Facts
;________________________________________________________________________________

(deffacts initial-flags

         (start-new-window-flag)          ; commence parametric regression process

         (retract-excess-data   TRUE)     ; any other value saves excess data

         (location              pool)     ; NPS swimming pool test environment

;;;;;;   (location              ocean)    ; alternative environment

)

;________________________________________________________________________________
;
; Global Constants
;________________________________________________________________________________

(defglobal ?*minimum_points_in_edge* = 5) ; hard coded into regression defrules

(defglobal ?*transducer_a* = 1); forward transducer corresponds to slot range_a
(defglobal ?*transducer_b* = 2); left    transducer corresponds to slot range_b
(defglobal ?*transducer_c* = 3); right   transducer corresponds to slot range_c
(defglobal ?*transducer_d* = 4); depth   transducer corresponds to slot range_d

(defglobal ?*feet_per_sonar_unit* = 0.02398) ; 0-4095 range units correspond to
                                             ;   0..30m pool or 0..300m ocean.

(defglobal ?*correlation_confidence_weight* = 1)
(defglobal ?*validity_confidence_weight*    = 1)

(defglobal ?*c1*        = 3.00)    ; max # standard deviations a point can be out
(defglobal ?*c2*        = 2.00)    ; max offset distance (feet) allowed from line
(defglobal ?*c3*        = 0.066)   ; min regression ellipse thinness requirement

; define default line/wall color strings
(defglobal ?*color1* = "Color      0  255  78 1.0   0 255  78"); detected edge
(defglobal ?*color2* = "Color    200  255 150 0.7 200 255 150"); inferred edge
(defglobal ?*color3* = "Color      0   78 255 0.5   0  78 255");  hidden  edge

(defglobal ?*min_wall_length*   = 1.0)    ; min allowable individual edge length
                                          ; to be output as a WALL

(defglobal ?*max_edge_distance* =  7.0)   ; max allowable distance between edges
                                          ; for edge-joining/polyhedron building

(defglobal ?*max_edge_angle*    = 10.0)   ; max allowable angle between edges
                                          ; for edge-joining/WALL building

(defglobal ?*wall_thinness_ratio* = 0.1) ; used to reclassify long skinny object
                                          ; as WALL

;________________________________________________________________________________
;
; Global Variables
;________________________________________________________________________________

(defglobal ?*n*        = 0.0)      ; m 00
(defglobal ?*sumx*     = 0.0)      ; m 10
(defglobal ?*sumy*     = 0.0)      ; m 01
(defglobal ?*sumxy*    = 0.0)      ; m 11
(defglobal ?*sumxx*    = 0.0)      ; m 20
(defglobal ?*sumyy*    = 0.0)      ; m 02
```

```
(defglobal ?*meanx*     = 0.0)        ; mu x
(defglobal ?*meany*     = 0.0)        ; mu y

(defglobal ?*sigmaxx*  = 0.0)        ; M 20
(defglobal ?*sigmaxy*  = 0.0)        ; M 11
(defglobal ?*sigmayy*  = 0.0)        ; M 02

(defglobal ?*phi*       = 0.0)        ; regression line orientation
(defglobal ?*r*         = 0.0)        ; regression line distance from origin

(defglobal ?*M-major*  = 0.0)        ; Moment around ellipse major axis
(defglobal ?*M-minor*  = 0.0)        ; Moment around ellipse minor axis

(defglobal ?*d-major*  = 0.0)        ; diameter on ellipse major axis
(defglobal ?*d-minor*  = 0.0)        ; diameter on ellipse minor axis

(defglobal ?*rho*       = 0.0)        ; ratio of major to minor axis diameters
(defglobal ?*delta*     = 0.0)        ; residual of a point
(defglobal ?*sigma*     = 0.0)        ; standard deviation

(defglobal ?*projection-x* = 0.0)    ; x projection of point i on major axis
(defglobal ?*projection-y* = 0.0)    ; y projection of point i on major axis

(defglobal ?*minx*      = 145)        ; for plot/graph boundaries
(defglobal ?*maxx*      = -15)        ;

(defglobal ?*miny*      = -50)        ; for plot/graph boundaries
(defglobal ?*maxy*      = 110)        ;

(defglobal ?*minz*      = 0.0)        ; for the currently active edge only
(defglobal ?*maxz*      = 0.0)        ; for the currently active edge only

(defglobal ?*defaultz* = 2.0)        ; default pool depth to be used for objects
                                      ;   of unspecified or indeterminate depth

(defglobal ?*offsetx*  = 0.0)        ; displacement added to (x, y, z) positional
(defglobal ?*offsety*  = 0.0)        ;    data to account for distance of the AUV
(defglobal ?*offsetz*  = 0.0)        ;    from the origin (i.e. corner) of the
                                      ;    NPS pool coordinate system

(defglobal ?*time*      = 0.0)        ; used to measure execution time

(defglobal ?*out*       = stdout)    ; verbose output default to stdout
                                      ;   otherwise ?*out* is reset to nil


(defglobal ?*gyroerror*          = 0.0)  ; User-provided gyro error (degrees)
(defglobal ?*newgyroerror*       = 0.0)  ; Expert system gyro error (degrees)
(defglobal ?*gyroerrortime*      = 0.0)  ; Average time of first wall found, used
                                          ;  as input to drift rate computation

(defglobal ?*gyrodriftrate*      = 0.0)  ; User-provided drift rate
(defglobal ?*newgyrodriftrate*   = 0.0)  ; Expert system drift rate

(defglobal ?*number_of_fields*  = 17)   ; read only actual # of input fields


;_______________________________________________________________________________

; A sample fact (for syntax training use only!)

(deffacts range1 (Range_data (time 0)
                             (x   2)  (y     3)  (z    4)
                             (phi 5)  (theta 6)  (psi  7)
                             (p   8)  (q     9)  (r   10)
                             (delta_dive_planes      11)
                             (delta_rudders          12)
                             (range_a 13)    (valid_a 1)
```

```
                                        (range_b 15)    (valid_b 1)
                                        (speed_17) (processed TRUE))
)

;_______________________________________________________________________
;
; Functions
;_______________________________________________________________________

;  atan2 function matches C language and class text syntax.
;  Calling order:  (atan2 y x)

(defmethod atan2 ((?y NUMBER)(?x NUMBER (> ?x 0)))
  (atan (/ ?y ?x)))

(defmethod atan2 ((?y NUMBER (> ?y 0)) (?x NUMBER (< ?x 0)))
  (+ (atan (/ ?y ?x)) (pi)))

(defmethod atan2 ((?y NUMBER (< ?y 0)) (?x NUMBER (< ?x 0)))
  (- (atan (/ ?y ?x)) (pi)))

(defmethod atan2 ((?y NUMBER (> ?y 0)) (?x NUMBER (= ?x 0)))
  (/ (pi) 2.0))

(defmethod atan2 ((?y NUMBER (< ?y 0)) (?x NUMBER (= ?x 0)))
  (/ (pi) -2.0))

(defmethod atan2 ((?y NUMBER (= ?y 0)) (?x NUMBER (< ?x 0)))
  (pi))

(defmethod atan2 ((?y NUMBER (= ?y 0)) (?x NUMBER (= ?x 0)))
  0.0)

;_______________________________________________________________________

(deffunction normalize (?x)  ; x in degrees, resulting range (0 .. 360)

   (bind ?norm ?x)
   (while (<  ?norm 0.0)   (bind ?norm (+ ?norm 360.0)))
   (while (>= ?norm 360.0) (bind ?norm (- ?norm 360.0)))
   ?norm
)

;_______________________________________________________________________

(deffunction normalize2 (?x)  ; x in degrees, resulting range (-180 .. 180)

   (bind ?norm ?x)
   (while (<  ?norm -180.0) (bind ?norm (+ ?norm 360.0)))
   (while (>= ?norm  180.0) (bind ?norm (- ?norm 360.0)))
   ?norm
)

;_______________________________________________________________________

(deffunction avg (?number1 ?number2)
   (/ (+ ?number1 ?number2) 2.0))

;_______________________________________________________________________

(deffunction degrees (?x)  ; x in radians

   (/ (* ?x 180.0) (pi))
)

;_______________________________________________________________________

(deffunction radians (?x)  ; x in radians
    (* (/ ?x 180.0) (pi))
)

;________________________________________________________________________________

; Boolean function to ask a yes/no question

(deffunction yes-or-no (?question)          ; '?question' is the question string

  (format t "%n%s?  " ?question) ; ask the question
  (bind ?answer (lowcase (sym-cat (read))))

  (while (and (neq ?answer yes)   (neq ?answer y)   (neq ?answer yep)
              (neq ?answer yeah)  (neq ?answer ye)  (neq ?answer yea)
              (neq ?answer no)    (neq ?answer n)
              (neq ?answer nope)  (neq ?answer nah))
         (format t "%n  Please answer yes or no:  ")
         (bind ?answer (lowcase (sym-cat (read)))))

  (if (or (eq ?answer yes)   (eq ?answer y)   (eq ?answer yep)
          (eq ?answer yeah)  (eq ?answer ye)  (eq ?answer yea))
      then TRUE
      else FALSE)
)

;________________________________________________________________________________

(deffunction distance  (?x1 ?y1 ?z1 ?x2 ?y2 ?z2)

   (sqrt (+ (*  (- ?x1 ?x2)  (- ?x1 ?x2))
            (*  (- ?y1 ?y2)  (- ?y1 ?y2))
            (*  (- ?z1 ?z2)  (- ?z1 ?z2))
    ))
)

;________________________________________________________________________________
;
; Triangle S and area calculation functions

;________________________________________________________________________________
;
(deffunction S     (?node1x ?node1y ?node2x ?node2y ?node3x ?node3y)

;   CCW triples are positive & CW triples are negative, matching conventions.

   (bind ?trianglearea
                           (* 0.5 (- (*  (- ?node2x ?node1x)  (- ?node3y ?node1y))
                                     (*  (- ?node3x ?node1x)  (- ?node2y ?node1y)))))
?trianglearea)

;________________________________________________________________________________
;
(deffunction area (?node1x ?node1y ?node2x ?node2y ?node3x ?node3y)

;   area values are always positive, matching conventions.

   (bind ?trianglearea
                      (abs (* 0.5 (- (*  (- ?node2x ?node1x)  (- ?node3y ?node1y))
                                     (*  (- ?node3x ?node1x)  (- ?node2y ?node1y))))))
?trianglearea)

;________________________________________________________________________________
;
; Expert system start and data file reading rules
;________________________________________________________________________________

(defrule get-initial-expert-system-parameters-and-open-range-file
    (declare (salience 100))
    (initial-fact)
=>
    (dribble-off)
    (system "mv -f auvsonar.log auvsonar.log.bak")
    (dribble-on auvsonar.log)

    (printout t crlf crlf "Name of range data file to open?  ")
    (bind ?filename (read))
    (open  ?filename rangefile "r")
    (printout t "Opened range data file " ?filename  crlf)

    (printout t crlf)
    (if   (yes-or-no "Are there more than 17 fields per Range_data record")
     then (printout t crlf "Enter number of data fields per record:  ")
          (bind ?*number_of_fields* (read))
          (while (or (< ?*number_of_fields* 17) (> ?*number_of_fields* 20))
                 (printout t crlf "Enter a value from 17..20:  ")
                 (bind ?*number_of_fields* (read))
                 (printout t crlf crlf)))

    ;     Determine output device for trace statements using 'format ?*out*'
    (printout t crlf)
    (if   (yes-or-no "Do you want verbose output onscreen during analysis")
     then (bind ?*out* stdout)
     else (bind ?*out* nil))

    (printout t crlf)
    (if   (yes-or-no "Do you want to input gyro error and gyro drift rate")
     then (printout t crlf "Enter gyro error       (degrees):  ")
          (bind ?*gyroerror* (normalize2 (read)))
          (printout t crlf "Enter gyro drift rate (degs/hr):  ")
          (bind ?*gyrodriftrate* (read)))
    (printout t crlf)

    (printout t crlf)
    (printout t crlf "Enter offset distance to be added to X positions to ")
    (printout t "account for the initial AUV displacement from pool corner:   ")
    (bind ?*offsetx* (read))
    (printout t crlf)

    (printout t crlf "Enter offset distance to be added to Y positions to ")
    (printout t "account for the initial AUV displacement from pool corner:   ")
    (bind ?*offsety* (read))
    (printout t crlf)

    (printout t crlf "Enter offset depth    to be added to Z positions to ")
    (printout t "account for the initial AUV displacement from pool surface:  ")
    (bind ?*offsetz* (read))
    (printout t crlf)

    (printout t crlf "Saving previous files pool.graph and pool.auv:" crlf)
    (printout t "mv -f  pool.auv    pool.auv.bak")
    (system     "mv -f  pool.auv    pool.auv.bak")
    (printout t crlf)
    (printout t "mv -f  pool.graph  pool.graph.bak")
    (system     "mv -f  pool.graph  pool.graph.bak")
    (printout t crlf)

    (open  "pool.auv"   auvfile  "a")
    (open  "pool.graph" plotfile "a")

    (printout auvfile crlf crlf
                      "         NPS AUV Sonar Classification Expert System"
                      "  (pool data " ?filename ")"
                      crlf crlf crlf)
    (printout auvfile crlf "All data values & type specifications are "
                    "defined by the AUV Data Dictionary."
               crlf)
  (printout auvfile crlf "All coordinate values are relative to the "
                    "NPS Pool Coordinate System."
               crlf crlf crlf)
  (printout auvfile crlf "AUV          "
                    ?*offsetx* "  " ?*offsety* "  " ?*offsetz*
            "  (xyz distances from AUV start position to pool origin)" crlf)

  (printout plotfile " 105.0  95.0 " crlf "\"NPS AUV Sonar Classification "
                "Expert System  (pool data " ?filename ") \" "
                crlf)

  (printout plotfile " 100.0 -30.0 " crlf "\"AUV start..origin offset values:  "
                                        ?*offsetx* ", "
                                        ?*offsety* ", "
                                        ?*offsetz*        " \" "
                crlf)

  (printout plotfile " 105.0 -40.0 " crlf "\"Parametric regression constants:  "
                                         "c1=" ?*c1*
                                        ", c2=" ?*c2*
                                        ", c3=" ?*c3*  " \" "
                crlf)

  (printout plotfile "   0.0   0.0 " crlf  ; pool boundary outline
                " 127.0   0.0 " crlf
                " 127.0  67.5 " crlf
                "   0.0  67.5 " crlf
                "   0.0   0.0 " crlf  " \" \" "
                crlf)

  (printout auvfile crlf "Environment    nps_pool.off"
               crlf "Replayfile     " ?filename
               crlf "Replaysize     " ?*number_of_fields*
               crlf) ; simulator replay filename/filesize initialization

  (if (or (<> ?*gyroerror* 0.0) (<> ?*gyrodriftrate* 0.0)) then
      (printout plotfile " 100.0 -20.0 " crlf "\"AUV gyro error = "
                    ?*gyroerror*      " degrees, gyro drift rate = "
                    ?*gyrodriftrate*  " degrees/hour \" "
                    crlf))

  (if (or (<> ?*gyroerror* 0.0) (<> ?*gyrodriftrate* 0.0)) then
      (printout auvfile "gyroerror      " ?*gyroerror*     " degrees" crlf
                   "gyrodriftrate  " ?*gyrodriftrate* " degrees/hour"
                    crlf))

  (printout auvfile crlf ?*color1* "  Color scheme for regression lines "
               crlf) ; primary default color scheme

  (bind ?*time* (time)) ; start clock timer

  (assert (check-file-flag))
)

;_______________________________________________________________________________

(defrule check-range-file

  ?check-file <- (check-file-flag)
  (not (range-file-closed-flag))
=>
  (retract ?check-file)  ; don't read this file again until point is processed

  (assert (first-element-read-file-flag = (read rangefile)))
  ; first-element-read-file-flag will be asserted with first element from
  ;       the rangefile
```

```
)

;_______________________________________________________________________________

(defrule skip-rangefile-comments  ; keep reading the file until we get a number

  (declare (salience 100))
  ?first-element-read-file <-(first-element-read-file-flag ?file-element & ~EOF)
  (test (not (numberp ?file-element)))
=>
  (retract ?first-element-read-file)
  (printout t ".")
  (readline rangefile)          ; flush comments through end-of-line
  (assert (check-file-flag))
)

;_______________________________________________________________________________

(defrule read-remainder-of-range-record

 (declare (salience 100))
 ?first-element-read-file <- (first-element-read-file-flag ?file-element & ~EOF)
 (test (numberp ?file-element))
=>
  (retract ?first-element-read-file)
  (bind ?field1 ?file-element)
  (bind ?field2  (read rangefile))
  (bind ?field3  (read rangefile))
  (bind ?field4  (read rangefile))
  (bind ?field5  (read rangefile))
  (bind ?field6  (read rangefile))
  (bind ?field7  (read rangefile))
  (bind ?field8  (read rangefile))
  (bind ?field9  (read rangefile))
  (bind ?field10 (read rangefile))
  (bind ?field11 (read rangefile))
  (bind ?field12 (read rangefile))
  (bind ?field13 (read rangefile))
  (bind ?field14 (read rangefile))
  (bind ?field15 (read rangefile))
  (bind ?field16 (read rangefile))
  (bind ?field17 (read rangefile))

  (if (>= ?*number_of_fields* 18) then (bind ?field18 (read rangefile)))
  (if (>= ?*number_of_fields* 19) then (bind ?field19 (read rangefile)))
  (if (>= ?*number_of_fields* 20) then (bind ?field20 (read rangefile)))

 ; account for user-provided gyro error and gyro drift rate:

 (bind ?totalerror (radians (+ ?*gyroerror*
                              (* ?*gyrodriftrate* (/ ?field1 3600.0)))))

  (bind ?heading (- ?field7 ?totalerror))

  ; Don't assert a point if it has no range value (non-return)
  (if (or (> ?field13 1) (> ?field14 1) (> ?field15 1) (> ?field16 1))
   then
       (assert (Range_data (time              ?field1)
                           (x                 ?field2)
                           (y                 ?field3)
                           (z                 ?field4)
                           (phi               ?field5)
                           (theta             ?field6)
                           (psi               ?heading)
                           (p                 ?field8)
                           (q                 ?field9)
                           (r                 ?field10)
                           (delta_dive_planes ?field11)
```

```
                              (delta_rudders      ?field12)
                              (range_a            ?field13)
                              (range_b            ?field14)
                              (range_c            ?field15)
                              (range_d            ?field16)
                              (speed              ?field17))))
  (format ?*out* "%nCompleted reading range record;  data time %3.1f" ?field1)
  (assert (check-file-flag))
)

;________________________________________________________________________________

(defrule close-range-file

  (declare (salience 100))
  ?first-element-read-file <- (first-element-read-file-flag EOF)
=>
  (retract ?first-element-read-file)
  (close rangefile)
  (assert (range-file-closed-flag))
  (assert (Point (status NEW)))  ; dummy point so last line (if any) is saved
  (printout t crlf "Closed the input range file." crlf)
)

;________________________________________________________________________________
;
; Point position calculation functions
;________________________________________________________________________________
;
; Forward transducer (#1):  reference frame is identical to AUV
; Left    transducer (#2):  psi   = AUV psi   + PI / 2
; Right   transducer (#3):  psi   = AUV psi   - PI / 2
; Depth   transducer (#4):  theta = AUV theta + PI / 2
;________________________________________________________________________________

(deffunction delta_x (?range ?phi ?theta ?psi)

  (if (= ?*transducer_b*  1)
     then (bind ?result  (* ?range (* (cos ?theta)  (cos ?psi)))))
  (if (= ?*transducer_b*  2)
     then (bind ?result  (* ?range (* (cos ?phi)   (cos (- ?psi (/ (pi) 2)))))))
  (if (= ?*transducer_b*  3)                             ;  | Note caveat about yaw
     then (bind ?result  (* ?range (* (cos ?phi)   (cos (+ ?psi (/ (pi) 2)))))))
  (if (= ?*transducer_b*  4)
     then (bind ?result  (* ?range (sin ?theta))))
  ?result)

;________________________________________________________________________________

(deffunction delta_y (?range ?phi ?theta ?psi)
  (if (= ?*transducer_b*  1)
     then (bind ?result  (* ?range (* (cos ?theta)  (sin ?psi)))))
  (if (= ?*transducer_b*  2)
     then (bind ?result  (* ?range (* (cos ?phi)   (sin (- ?psi (/ (pi) 2)))))))
  (if (= ?*transducer_b*  3)                             ;  | Note caveat about yaw
     then (bind ?result  (* ?range (* (cos ?phi)   (sin (+ ?psi (/ (pi) 2)))))))
  (if (= ?*transducer_b*  4)
     then (bind ?result  (* ?range (sin ?phi))))
  ?result)

;________________________________________________________________________________

(deffunction delta_z (?range ?phi ?theta ?psi)

  (if (= ?*transducer_b*  1)
     then (bind ?result  (* ?range (sin ?theta))))
  (if (= ?*transducer_b*  2)
     then (bind ?result  (- 0 (* ?range (sin ?phi)))))
  (if (= ?*transducer_b*  3)
      then (bind ?result  (* ?range (sin ?phi))))
  (if (= ?*transducer_b*  4)
      then (bind ?result  (* ?range (* (cos ?phi) (cos ?theta)))))
  ?result)

;________________________________________________________________________________
;
;  Point building rule
;________________________________________________________________________________

(defrule build-point-from-raw-AUV-range-data

; this rule currently handles only left transducer

  (declare (salience 200))
  ?range_data<- (Range_data (processed FALSE)
                            (time ?time) (x ?x) (y ?y) (z ?z)
                            (phi ?phi) (theta ?theta) (psi ?psi)
                            (range_b ?range)(valid_b ?valid))
  (test (<> ?*transducer_b* 0))
=>
  (bind ?range (* ?range ?*feet_per_sonar_unit*)); unit conversion of range slot
  (bind ?delta_x (delta_x ?range ?phi ?theta ?psi))
  (bind ?delta_y (delta_y ?range ?phi ?theta ?psi))
  (bind ?delta_z (delta_z ?range ?phi ?theta ?psi))
  (if (and (> ?time 0) (> ?range 1)) then ; only make valid data points

     (assert (Point (time ?time)
                    (x =(+ ?x  ?delta_x))
                    (y =(+ ?y  ?delta_y))
                    (z =(+ ?z  ?delta_z))
                    (valid ?valid)
                    (status NEW)))

;     print sonar return as 'o' and auv position as '*'
     (printout plotfile (+ ?x ?delta_x ?*offsetx*)      " "
                        (+ ?y ?delta_y ?*offsety*) crlf "o" crlf)
     (printout plotfile (+ ?x          ?*offsetx*)      " "
                        (+ ?y          ?*offsety*) crlf "*" crlf)
                      ; include coordinate offsets
     (modify ?range_data (processed TRUE) (range_b ?range))
     (format ?*out* "%nAsserted and plotted a point for data time %3.1f" ?time))
  else  ; a bogus point
     (modify ?range_data (processed TRUE) (range_b ?range))
)

;________________________________________________________________________________
;
; Two-dimensional parametric regression line analysis rules
;________________________________________________________________________________

(defrule regression-line-sliding-window-start-criteria

  (declare (salience 300))
  ?start-new-window <- (start-new-window-flag)
  ; Find the next 5 NEW points
  ?point1 <- (Point (status NEW) (time ?time1)(x ?x1)(y ?y1)(z ?z1))
  ?point2 <- (Point (status NEW) (time ?time2)(x ?x2)(y ?y2)(z ?z2))
  ?point3 <- (Point (status NEW) (time ?time3)(x ?x3)(y ?y3)(z ?z3))
  ?point4 <- (Point (status NEW) (time ?time4)(x ?x4)(y ?y4)(z ?z4))
  ?point5 <- (Point (status NEW) (time ?time5)(x ?x5)(y ?y5)(z ?z5))
  (test (< ?time1 ?time2))
  (test (< ?time2 ?time3))
  (test (< ?time3 ?time4))
  (test (< ?time4 ?time5))
=>
  (retract ?start-new-window)
  ; These points are eligible and thus become ACTIVE
  (modify ?point1 (status ACTIVE))
  (modify ?point2 (status ACTIVE))
  (modify ?point3 (status ACTIVE))
  (modify ?point4 (status ACTIVE))
  (modify ?point5 (status ACTIVE))
  (bind ?*n*      5)
  (bind ?*sumx*  (+ ?x1 ?x2 ?x3 ?x4 ?x5))
  (bind ?*sumy*  (+ ?y1 ?y2 ?y3 ?y4 ?y5))
  (bind ?*sumxy* (+ (* ?x1 ?y1) (* ?x2 ?y2)(* ?x3 ?y3)(* ?x4 ?y4)(* ?x5 ?y5)))
  (bind ?*sumxx* (+ (* ?x1 ?x1) (* ?x2 ?x2)(* ?x3 ?x3)(* ?x4 ?x4)(* ?x5 ?x5)))
  (bind ?*sumyy* (+ (* ?y1 ?y1) (* ?y2 ?y2)(* ?y3 ?y3)(* ?y4 ?y4)(* ?y5 ?y5)))
  (bind ?*minz*  (min ?z1 ?z2 ?z3 ?z4 ?z5))
  (bind ?*maxz*  (max ?z1 ?z2 ?z3 ?z4 ?z5))

  (assert (Regression_line (start ?time1) (end ?time5) (status NEW)))
  (format ?*out* "%n%nRegression line sliding window start criteria met.")
)

;__________________________________________________________________________

(deffunction calculate-line-fit-and-update-global-variables ()

;  global inputs:  n, sumx, sumy, sumxy, sumxx, sumyy

   (bind ?*meanx* (/ ?*sumx* ?*n*))
   (bind ?*meany* (/ ?*sumy* ?*n*))

   (bind ?*sigmaxx* (- ?*sumxx* (/ (* ?*sumx* ?*sumx*) ?*n*)))
   (bind ?*sigmaxy* (- ?*sumxy* (/ (* ?*sumx* ?*sumy*) ?*n*)))
   (bind ?*sigmayy* (- ?*sumyy* (/ (* ?*sumy* ?*sumy*) ?*n*)))

   (bind ?*phi*    (* 0.5 (atan2 (* -2.0 ?*sigmaxy*) (- ?*sigmayy* ?*sigmaxx*))
                    )) ; note paper's caveat re frame of reference of phi

   (bind ?*r*       (+ (* ?*meanx* (cos ?*phi*)) (* ?*meany* (sin ?*phi*))))

   (bind ?term2     (sqrt (+ (* 0.25 (- ?*sigmayy* ?*sigmaxx*)
                                     (- ?*sigmayy* ?*sigmaxx*))
                             (* ?*sigmaxy* ?*sigmaxy*))))

   (bind ?*M-major* (- (/ (+ ?*sigmaxx* ?*sigmayy*) 2.0) ?term2))
   (bind ?*M-minor* (+ (/ (+ ?*sigmaxx* ?*sigmayy*) 2.0) ?term2))

   (bind ?*d-major* (* 4 (sqrt (/ ?*M-minor* ?*n*))))
   (bind ?*d-minor* (* 4 (sqrt (/ ?*M-major* ?*n*))))

   (bind ?*rho*     (/ ?*d-minor* ?*d-major*))
   (format ?*out* "%nRegression line fit calculations complete.")
)

;__________________________________________________________________________

(defrule regression-line-initial-segment-validity-check

   (declare (salience 300))
   ; Get the NEW Regression_line and 5 ACTIVE Points
   ?line   <- (Regression_line (start ?time1) (end ?time5) (status NEW))

   ?point1 <- (Point (time ?time1) (x ?x1) (y ?y1) (z ?z1) (valid ?valid1))
   ?point5 <- (Point (time ?time5) (x ?x5) (y ?y5) (z ?z5) (valid ?valid5))

   ?point2 <- (Point (time ?time2) (x ?x2) (y ?y2) (z ?z2) (valid ?valid2))
   (test (and (< ?time1 ?time2) (> ?time5 ?time2)))
   ?point3 <- (Point (time ?time3) (x ?x3) (y ?y3) (z ?z3) (valid ?valid3))
   (test (and (< ?time1 ?time3) (> ?time5 ?time3) (<> ?time2 ?time3)))
   ?point4 <- (Point (time ?time4) (x ?x4) (y ?y4) (z ?z4) (valid ?valid4))
   (test (and (< ?time1 ?time4) (> ?time5 ?time4) (<> ?time2 ?time4)
              (<> ?time3 ?time4)))
=>
   (calculate-line-fit-and-update-global-variables)

   (bind ?*rho* (/ ?*d-minor* ?*d-major*))

   (if   (< ?*rho* ?*c3*) ; Validity check:  Test II equation (25)

    then                  ; initial line segment IS valid
         (modify ?point1 (status ENDPOINT))
         (modify ?point2 (status USED))
         (modify ?point3 (status USED))
         (modify ?point4 (status USED))
         (modify ?point5 (status ENDPOINT))
         (modify ?line   (status CURRENT)
                         (r ?*r*)
                         (orientation =(normalize (degrees
                                             (atan2 (- ?y5 ?y1) (- ?x5 ?x1)))))
                         (correlation ?*rho*))
      (format ?*out* "%nRegression line initial segment validity check passed.")

    else                  ; initial line segment IS NOT valid
         (modify ?point1 (status INVALID))  ; window slides by one to the right
         (modify ?point2 (status NEW))
         (modify ?point3 (status NEW))
         (modify ?point4 (status NEW))
         (modify ?point5 (status NEW))
         (retract ?line)
         (assert (start-new-window-flag))   ; begin building a new window
      (format ?*out* "%nRegression line initial segment validity check failure.")
      (format ?*out* "%n")
   )
)

;________________________________________________________________________________

(defrule regression-line-window-expansion

   (declare (salience 300))
   ; Get the CURRENT Regression_line, start  Point, end Point, and new Point
   ?current-line <- (Regression_line (start ?starttime) (end ?endtime)
                                     (status CURRENT))
   ?new-point    <- (Point (time ?newtime) (x ?newx) (y ?newy)(z ?newz)
                           (status NEW))
   ?start-point  <- (Point (time ?starttime)(x ?startx)(y ?starty)(z ?startz))
   ?end-point    <- (Point (time ?endtime)  (x ?endx)  (y ?endy)  (z ?endz))
=>
  (bind ?*delta* (+ (* (cos ?*phi*)(- ?*meanx* ?newx))
                    (* (sin ?*phi*)(- ?*meany* ?newy)))) ; residual

  (bind ?*sigma* (sqrt (/ ?*M-minor* (- ?*n* 2))))

  (if (and (< ?*delta* (max (* ?*c1* ?*sigma*) ?*c2*)) ; Test I  equation (23)
           (< ?*rho*    ?*c3*)                         ; Test II equation (25)
           (> ?newtime  0))                            ; ignore invalid points


    then                                 ;test passed, new point meets criteria
      (modify ?new-point (status USED))  ; we just used this point
      (bind ?*n*     (+ ?*n* 1))
      (bind ?*sumx*  (+ ?*sumx* ?newx))
      (bind ?*sumy*  (+ ?*sumy* ?newy))
      (bind ?*sumxy* (+ ?*sumxy* (* ?newx ?newy)))
      (bind ?*sumxx* (+ ?*sumxx* (* ?newx ?newx)))
      (bind ?*sumyy* (+ ?*sumyy* (* ?newy ?newy)))
      (bind ?*minz*  (min ?*minz* ?newz))
      (bind ?*maxz*  (max ?*maxz* ?newz))
    ;update globals and then line parameters
    (calculate-line-fit-and-update-global-variables)

    (bind ?correlation     (- 1 ?*rho*))

    ; update endpoint status slots for possible retraction of used data
    (modify ?end-point     (status USED))
    (modify ?new-point     (status ENDPOINT))

    (modify ?current-line (r    ?*r*)
            (orientation =(normalize (degrees
                                (atan2 (- ?newy ?starty) (- ?newx ?startx)))))
                          (correlation ?correlation) ; value range [1-c3..1]
                          (end ?newtime))
    (format ?*out* "       Added another point to the regression line.%n");

  else
    (modify ?current-line (status VALID))  ; test failed, save old line

    ;current point retains status NEW unless it is a dummy point at time zero
    (if (= 0 ?newtime) then (retract ?new-point))

    ; initial node of new segment

    (bind ?*delta* (+ (* (cos ?*phi*)(- ?*meanx* ?startx))
                      (* (sin ?*phi*)(- ?*meany* ?starty))))  ; residual

    (bind ?*projection-x*      (+ ?startx (* ?*delta* (cos ?*phi*))))
    (bind ?*projection-y*      (+ ?starty (* ?*delta* (sin ?*phi*))))
    (bind ?start-projection-x  ?*projection-x*)
    (bind ?start-projection-y  ?*projection-y*)
    (bind ?correlation         (- 1 ?*rho*))
    (assert (Node (time ?starttime)           ; edge's virtual start node
                  (x          ?*projection-x*)
                  (y          ?*projection-y*)
                  (z          ?startz)
                  (accuracy   ?*d-minor*)      ; minor axis diameter
                  (confidence ?correlation))) ; using elliptical thinness
    (format ?*out* "  Valid node completed, data time %3.1f%n" ?starttime)

    (printout plotfile (+ ?*projection-x* ?*offsetx*)  " "
                       (+ ?*projection-y* ?*offsety*)  crlf)
    (format ?*out* "  Projection endpoints (%5.1f, %5.1f)"
                   (+ ?*projection-x* ?*offsetx*)
                   (+ ?*projection-y* ?*offsety*))

    ; final node of new segment

    (bind ?*delta* (+ (* (cos ?*phi*)(- ?*meanx* ?endx))
                      (* (sin ?*phi*)(- ?*meany* ?endy))))  ; residual

    (bind ?*projection-x* (+ ?endx (* ?*delta* (cos ?*phi*))))
    (bind ?*projection-y* (+ ?endy (* ?*delta* (sin ?*phi*))))
    (bind ?confidence     (- 1 ?*rho*))
    (assert (Node (time       ?endtime)        ; edge's virtual end node
                  (x          ?*projection-x*)
                  (y          ?*projection-y*)
                  (z          ?endz)
                  (accuracy   ?*d-minor*)      ; minor axis diameter
                  (confidence ?confidence)))   ; using elliptical thinness
    (format ?*out* "  (%5.1f, %5.1f)%n"
                   (+ ?*projection-x* ?*offsetx*)
                   (+ ?*projection-y* ?*offsety*))

    (format ?*out* "  Raw data endpoints   (%5.1f, %5.1f)  (%5.1f, %5.1f)%n"
                   (+ ?startx  ?*offsetx*)
                   (+ ?starty  ?*offsety*)
                   (+ ?endx    ?*offsetx*)
```

```
                        (+ ?endy    ?*offsety*))
        (format ?*out* "  Valid node completed, data time %3.1f %n" ?endtime)
        (printout plotfile (+ ?*projection-x* ?*offsetx*)  " "
                            (+ ?*projection-y* ?*offsety*)  crlf "\" \"" crlf)

        (assert (Edge (start      ?starttime)
                      (end        ?endtime)
                      (averagez   =(avg ?*minz* ?*maxz*)))))
        (format ?*out* "  Valid edge completed, data times (%3.1f .. %3.1f),"
                        ?starttime  ?endtime)
        (format ?*out* " averagez = %3.1f, line r = %3.1f,"
                        (avg ?*minz* ?*maxz*) ?*r*)

        (format ?*out*  " line orientation = %3.1f degrees%n"
                (normalize (degrees (atan2 (- ?newy ?starty) (- ?newx ?startx)))))

        (format auvfile
             "%nPoint   %5.1f %4.1f %3.1f                   time %4.1f"
                         (+ ?start-projection-x ?*offsetx*)
                         (+ ?start-projection-y ?*offsety*)
                         (+ ?startz             ?*offsetz*)
                          ?starttime)
                          ; depth range 0..8 ft, time is optional

        (format auvfile
             "%nPoint   %5.1f %4.1f %3.1f                   time %4.1f"
                         (+ ?*projection-x* ?*offsetx*)
                         (+ ?*projection-y* ?*offsety*)
                         (+ ?endz           ?*offsetz*)
                          ?endtime)

        (format auvfile
             "%nSegment %5.1f %4.1f %3.1f  %5.1f %4.1f %3.1f  time %4.1f"
                         (+ ?start-projection-x   ?*offsetx*)
                         (+ ?start-projection-y   ?*offsety*)
                         (+ (/ (+ ?*minz* ?*maxz*) 2.0) ?*offsetz*)
                         (+ ?*projection-x*       ?*offsetx*)
                         (+ ?*projection-y*       ?*offsety*)
                         (+ (/ (+ ?*minz* ?*maxz*) 2.0) ?*offsetz*)
                          ?endtime)

       (assert (check-file-flag))
       (assert (start-new-window-flag))
       (format ?*out* "  Valid regression line actions completed, data time %3.1f"
                       ?endtime)
       (format ?*out* "%n%n")
      )
)

;_______________________________________________________________________________
;
; Rules for retraction of excess data facts (garbage collection)
;_______________________________________________________________________________

(defrule retract-excess-Range_data

   (retract-excess-data TRUE)
   ?range_data <- (Range_data (processed TRUE))
=>
   (retract ?range_data)
)

(defrule retract-excess-Point

   (retract-excess-data TRUE)
   ?point <- (Point (status INVALID | USED))
=>
   (retract ?point)
```

```
)

(defrule retract-excess-endPoint

   (retract-excess-data TRUE)
   ?point <- (Point (status ENDPOINT) (time   ?point-time))
   ?node  <- (Node                    (time   ?node-time))
   (test (= ?point-time ?node-time))
=>
   (retract ?point)
)

;____________________________________________________________________
;
; Gyro error rules
;____________________________________________________________________

(defrule determine-initial-gyro-error

   (declare (salience 300))
   ?pool <- (location pool) ; this rule only works in the pool
   ?poly <- (Polyhedron (classification WALL) (start ?polystart) (end ?polyend)
                        (status COMPLETE))
   (test (= ?*newgyroerror* 0.0)) ; first wall provides best est, don't repeat
   ?line <- (Regression_line (start ?start)(end ?end)(orientation ?orientation)
                        (status USED | USED_FOR_AREA))
   (test (= ?polyend ?end))

   ?point1 <- (Point (time ?time1) (x ?x1) (y ?y1) (z ?z1))
   (test (= ?time1 ?start))
   ?point2 <- (Point (time ?time2) (x ?x2) (y ?y2) (z ?z2))
   (test (= ?time2 ?end))
   (test (>= (distance ?x1 ?y1 ?z1 ?x2 ?y2 ?z2) 2.0)) ; skip short segments
=>
   (bind ?delta1 (normalize2 (- ?orientation   0.0)))
   (bind ?delta2 (normalize2 (- ?orientation  90.0)))
   (bind ?delta3 (normalize2 (- ?orientation 180.0)))
   (bind ?delta4 (normalize2 (- ?orientation 270.0)))

   (if (< (abs ?delta1) (min (abs ?delta2) (abs ?delta3) (abs ?delta4))) then
       (bind ?*newgyroerror* ?delta1))
   (if (< (abs ?delta2) (min (abs ?delta1) (abs ?delta3) (abs ?delta4))) then
       (bind ?*newgyroerror* ?delta2))
   (if (< (abs ?delta3) (min (abs ?delta2) (abs ?delta1) (abs ?delta4))) then
       (bind ?*newgyroerror* ?delta3))
   (if (< (abs ?delta4) (min (abs ?delta2) (abs ?delta3) (abs ?delta1))) then
       (bind ?*newgyroerror* ?delta4))

   (bind ?*gyroerrortime* (avg ?start ?end)) ; average time of wall segment
   (format t "%nUser-provided gyro error = %4.1f degrees" ?*gyroerror*)
   (format t "%nWall orientation         = %4.1f degrees" ?orientation)
   (format t " for time %3.1f (%3.1f .. %3.1f)"
              (avg ?start ?end)  ?start  ?end)
   (format t "%nExpert system gyro error = %4.1f degrees" ?*newgyroerror*)
)

;____________________________________________________________________

(defrule determine-gyro-drift-rate

   (declare (salience 300))
   ?pool <- (location pool) ; this rule only works in the pool
   ?poly <- (Polyhedron (classification WALL) (start ?polystart) (end ?polyend)
                        (status        COMPLETE))
   (test (<> ?*newgyroerror* 0.0)) ; perform only if new gyro error calculated
   ?line <- (Regression_line (start ?start)(end ?end)(orientation ?orientation)
                             (status USED | USED_FOR_AREA))
   (test (= ?polyend ?end))
```

```
=>
   (bind ?delta1 (normalize2 (- ?orientation ?*newgyroerror*  0.0 )))
   (bind ?delta2 (normalize2 (- ?orientation ?*newgyroerror*  90.0)))
   (bind ?delta3 (normalize2 (- ?orientation ?*newgyroerror* 180.0)))
   (bind ?delta4 (normalize2 (- ?orientation ?*newgyroerror* 270.0)))

   (if (< (abs ?delta1) (min (abs ?delta2) (abs ?delta3) (abs ?delta4))) then
       (bind ?*newgyrodriftrate* ?delta1))
   (if (< (abs ?delta2) (min (abs ?delta1) (abs ?delta3) (abs ?delta4))) then
       (bind ?*newgyrodriftrate* ?delta2))
   (if (< (abs ?delta3) (min (abs ?delta2) (abs ?delta1) (abs ?delta4))) then
       (bind ?*newgyrodriftrate* ?delta3))
   (if (< (abs ?delta4) (min (abs ?delta2) (abs ?delta3) (abs ?delta1))) then
       (bind ?*newgyrodriftrate* ?delta4))

   (format t "%nWall orientation              = %4.1f degrees" ?orientation)
   (format t " for time %3.1f (%3.1f .. %3.1f)"
             (avg ?start ?end)  ?start  ?end)
   (format t "%nCurrent gyro error            = %4.1f degrees%n"
             (+ ?*newgyrodriftrate* ?*newgyroerror*))
   (bind ?*newgyrodriftrate* (* 3600.0 (/ ?*newgyrodriftrate*
                                  (- (avg ?start ?end) ?*gyroerrortime*))))

   (if (<= (abs ?*newgyrodriftrate*) 200.0) then
       (format t "Expert system gyro drift rate = %4.1f degrees/hour %n"
                 ?*newgyrodriftrate*))
)

;______________________________________________________________________________
;
; Completion!
;______________________________________________________________________________

(defrule plot-pool-graph-file-when-done  ; this rule is the last to be fired

   (declare (salience 0))              ; all other rules take precedence
   ?range-file-closed <- (range-file-closed-flag)
=>
   (format t
       "%n%nElapsed time to perform sonar classification:  %3.1f seconds.%n%n"
       (- (time) ?*time*))

   ; all file outputs complete
   (close plotfile)
   (close auvfile)

   (printout t crlf crlf "Sending pool.auv to iris graphics subdirectory."
             crlf)
   ; first save old copy of file to pool.bak
   (printout t crlf "rcp gemini:~brutzman/clips/pool.auv.bak"
                     " iris1:~brutzman/graphics/pool.auv.bak"  crlf)
   (system          "rcp gemini:~brutzman/clips/pool.auv.bak"
                     " iris1:~brutzman/graphics/pool.auv.bak")

   (printout t crlf "rcp gemini:~brutzman/clips/pool.auv"
                     " iris1:~brutzman/graphics/pool.auv"  crlf)
   (system          "rcp gemini:~brutzman/clips/pool.auv"
                     " iris1:~brutzman/graphics/pool.auv")

   (printout t crlf "rcp gemini:~brutzman/clips/pool.auv"
                      " iris1:~brutzman/preview/pool.auv"  crlf)
   (system         "rcp  gemini:~brutzman/clips/pool.auv"
                      " iris1:~brutzman/preview/pool.auv")

    ; You must be running under sunview on a workstation for sunplot to work.
    (printout t crlf crlf "The generated pool.graph sunplot follows:" crlf crlf)

    (printout t "graph -b -g 1 -l \"NPS AUV Sonar Classification Expert System "
                "\" -x 145 -15 -y -50 110  < pool.graph | sunplot -c 650"
                crlf crlf)
      (system    "graph -b -g 1 -l \"NPS AUV Sonar Classification Expert System "
                 "\" -x 145 -15 -y -50 110  < pool.graph | sunplot -c 650")
      (system    "rm core") ; remove core dump file which resulted
                            ;   if not running under sunview

   (if   (yes-or-no " Do you want to print the screen log file")
    then (dribble-off)
         (system "enscript -G -r auvsonar.log"))

   (if   (yes-or-no "Do you want a hard copy of the sonar plot")
    then
      (open  "pool.auv"   auvfile  "a")
      (open  "pool.graph" plotfile "a")
      (printout t crlf)
      (if   (yes-or-no "Do you want to add a comment line to the plot")
       then (printout t crlf crlf "Enter comment:  ")
            (bind ?comments (readline))
            (printout   auvfile crlf "Comment:  " ?comments crlf)
            (printout  plotfile " 115 80 " crlf "\"" ?comments " \" " crlf))

      (printout t crlf crlf "The generated pool.graph is being plotted:"
                 crlf crlf)
      (printout t "graph -b -g 1 -l \"NPS AUV Sonar Classification Expert System "
                 "\" -x 145 -15 -y -50 110  < pool.graph | lpr -g -h -Pap2"
                 crlf crlf)
      (system    "graph -b -g 1 -l \"NPS AUV Sonar Classification Expert System "
                 "\" -x 145 -15 -y -50 110  < pool.graph | lpr -g -h -Pap2")
     ; all file outputs complete
      (close plotfile)
      (close auvfile)
   )

   (printout t crlf "The generated pool.auv file follows:" crlf crlf)
   (system "more pool.auv")
   (printout t crlf crlf crlf)
)

;________________________________________________________________________
;
; Polyhedron output function and rules
;________________________________________________________________________

(deffunction output_polyhedron (?starttime ?endtime
                                ?startx    ?starty    ?startz
                                ?endx      ?endy      ?endz
                                ?classification ?comment)

   (format   t "%n%nThe polyhedron at times (%3.1f .. %3.1f) "
                ?starttime ?endtime)
   (printout t "has classification " ?classification "." crlf crlf)

   (format auvfile
         "%n%s    %5.1f %4.1f %3.1f  %5.1f %4.1f %3.1f  time %4.1f  %s"
                   ?classification
                   (+ ?startx ?*offsetx*)
                   (+ ?starty ?*offsety*)
                   (+ ?startz ?*offsetz*)
                   (+   ?endx ?*offsetx*)
                   (+   ?endy ?*offsety*)
                   (+   ?endz ?*offsetz*)
                   ?endtime
                   ?comment)

     (format auvfile "%n")

     (format ?*out* "%n%n")
     (format ?*out*
          "%n%s    %5.1f %4.1f %3.1f  %5.1f %4.1f %3.1f  time %4.1f  %s"
                   ?classification
                   (+ ?startx ?*offsetx*)
                   (+ ?starty ?*offsety*)
                   (+ ?startz ?*offsetz*)
                   (+   ?endx ?*offsetx*)
                   (+   ?endy ?*offsety*)
                   (+   ?endz ?*offsetz*)
                   ?endtime
                   ?comment)

     (format ?*out* "%n%n")

)

;________________________________________________________________________

(defrule change-colors-for-inferred-edges-when-done

   (declare (salience 40))             ; pre-completion rules take precedence
   ?range-file-closed <- (range-file-closed-flag)
=>
   (printout auvfile crlf crlf ?*color2* "  Color scheme for inferred edges "
                      crlf) ; secondary default color scheme
)

;________________________________________________________________________

(defrule output-polyhedrons-with-inferred-edges-when-done

   (declare (salience 30))             ; pre-completion rules take precedence
   ?range-file-closed <- (range-file-closed-flag)
   ?poly <- (Polyhedron (status COMPLETE | USED_FOR_AREA)
                        (start    ?startpolytime)
                        (end      ?endpolytime)
                        (startx   ?startx)    (starty ?starty) (startz ?startz)
                        (trait    INFERRED_EDGE)
                        (classification WALL))
;   node matches end of polyhedron
   ?node <- (Node (time  ?nodetime) (x ?nodex) (y ?nodey) (z ?nodez))
   (test (= ?endpolytime ?nodetime))

=>
   (output_polyhedron  ?startpolytime ?endpolytime
                       ?startx ?starty 0.0
                       ?nodex  ?nodey  8.0
                       "WALL"
                       "(inferred edge)")
)

;________________________________________________________________________

(defrule change-colors-for-hidden-edges-when-done

   (declare (salience 20))             ; pre-completion rules take precedence
   ?range-file-closed <- (range-file-closed-flag)
=>
   (printout auvfile crlf crlf ?*color3* "  Color scheme for hidden edges "
                      crlf) ; tertiary default color scheme
)

;________________________________________________________________________

(defrule output-object-polyhedrons-with-hidden-edges-when-done

   (declare (salience 10))             ; pre-completion rules take precedence
   ?range-file-closed <- (range-file-closed-flag)
   ?poly <- (Polyhedron (status COMPLETE | USED_FOR_AREA)
                        (start   ?startpolytime)
                        (end     ?endpolytime)
                        (startx  ?startx)     (starty ?starty) (startz ?startz)
                        (trait   HIDDEN_EDGE)
                        (classification WALL))
;   node matches end of polyhedron
   ?node <- (Node (time  ?nodetime) (x ?nodex) (y ?nodey) (z ?nodez))
   (test (= ?endpolytime ?nodetime))

=>
   (output_polyhedron  ?startpolytime ?endpolytime
                       ?startx ?starty 0.0
                       ?nodex  ?nodey  8.0
                       "WALL"
                       "(hidden edge)")
)

;_______________________________________________________________________
;
; Polyhedron building rules
;_______________________________________________________________________

(defrule polyhedron-building-start

   (declare (salience 430)) ; lower salience value than polyhedron-building
   ?line <- (Regression_line (status VALID)
                             (start ?starttime) (end ?endtime))

   ?start-node <- (Node      (time     ?nodetime)
                             (accuracy ?accuracy1)
                             (x ?startx) (y ?starty) (z ?startz))
   (test (= ?starttime ?nodetime))

   ?end-node   <- (Node      (time     ?endnodetime)
                             (accuracy ?accuracy2)
                             (x ?endx) (y ?endy) (z ?endz))
   (test (= ?endtime ?endnodetime))
=>
   (assert (Polyhedron (status  ACTIVE)
                       (start  ?starttime) (end    ?endtime)
                       (startx ?startx)    (starty ?starty) (startz ?startz)
                       (centroidx  =(+ ?startx ?endx))
                       (centroidy  =(+ ?starty ?endy))
                       (centroidz  =(+ ?startz ?endz))
                       (sidecount  1)
                       (accuracy   =(max ?accuracy1 ?accuracy2))
                       (trait      OBJECT_BUILDING_BASED)
                       (classification WALL)))

   (modify ?line       (status         USED))

   (bind ?length (distance ?startx ?starty ?startz ?endx ?endy ?endz))

   (if   (>= ?length ?*min_wall_length*)
    then (output_polyhedron  ?starttime ?endtime
                             ?startx ?starty 0.0
                             ?endx   ?endy   8.0
                              WALL "(new polyhedron start)"))

)

;_______________________________________________________________________

(defrule polyhedron-building-continuation

;  This rule tests the newest edge in relation to the most recent previous edge
;     of the currently ACTIVE polyhedron.
```

```
;
;  If the edges are too far apart, the previous polyhedron is COMPLETE and the
;     new edge is ignored in order for it to begin a new polyhedron.
;
;  If the edges are colinear, the new edge is included as part of the
;     currently ACTIVE polyhedron.

;  If the edges are concave, the previous polyhedron is COMPLETE and the new
;     edge is ignored in order to let it begin a new polyhedron.
;
;  If the edges are convex, the new edge is included as part of the
;     currently ACTIVE polyhedron, and the polyhedron is reclassified
;     from WALL to OBJECT.
;
; Specific polyhedron OBJECT reclassifications will be made by higher level rules.

   (declare (salience 440)) ; higher salience value than polyhedron start
   ?poly <- (Polyhedron (status      ACTIVE)
                        (start      ?startpolytime)
                        (end        ?endpolytime)
                        (accuracy   ?polyaccuracy)
                        (startx     ?startx)
                        (starty     ?starty)
                        (startz     ?startz)
                        (centroidx  ?centroidx)
                        (centroidy  ?centroidy)
                        (centroidz  ?centroidz)
                        (sidecount  ?sidecount)
                        (area       ?area)
                        (classification     ?classification))

;   line1 is most recent valid regression line included in the polyhedron
   ?line1 <- (Regression_line (status USED)
                              (start ?startline1time)
                              (end     ?endline1time)
                              (orientation ?orientation1))
   (test (= ?endpolytime ?endline1time))

;   line2 is newest valid regression line to be evaluated
   ?line2 <- (Regression_line (status VALID)
                              (start ?startline2time)
                              (end     ?endline2time)
                              (orientation ?orientation2))

;   node1 matches end of line1 (most recent valid regression line)
   ?node1 <- (Node (time  ?node1time)
                   (accuracy ?accuracy2)
                   (x ?node1x) (y ?node1y) (z ?node1z))
   (test (= ?endline1time ?node1time))

;   node2 matches start of line2
   ?node2 <- (Node (time  ?node2time)
                   (accuracy ?accuracy1)
                   (x ?node2x) (y ?node2y) (z ?node2z))
   (test (= ?startline2time ?node2time))

;   node3 matches end of line2
   ?node3 <- (Node (time  ?node3time)
                   (accuracy ?accuracy3)
                   (x ?node3x) (y ?node3y) (z ?node3z))
   (test (= ?endline2time ?node3time))

=>
   (bind ?distance (distance ?node1x ?node1y ?node1z ?node2x ?node2y ?node2z))

;   if distance is too great, don't continue building polyhedron with new edge
   (if    (>  ?distance ?*max_edge_distance*)
    then (modify ?poly (status COMPLETE)
                              (area        =(abs ?area))
                              (centroidx   =(/ ?centroidx ?sidecount 2)) ; 2 points/side
                              (centroidy   =(/ ?centroidy ?sidecount 2))
                              (centroidz   =(/ ?centroidz ?sidecount 2))
                              (sidecounter1    ?sidecount)
                              (sidecounter2    ?sidecount))

                  ; if polyhedron was not a WALL, assert a HIDDEN_EDGE wall for it
                   (if    (not (eq ?classification WALL))
                    then (format t "%nPolyhedron OBJECT (%3.1f .. %3.1f) "
                                             ?startpolytime ?endpolytime)
                         (format t "has area %3.1f%n" (abs ?area))
                         (format auvfile "  (prior object area was %3.1f)" (abs ?area))
                         (assert (Polyhedron (status        COMPLETE)
                                             (start        ?startpolytime)
                                             (end          ?endpolytime)
                                             (startx       ?startx)
                                             (starty       ?starty)
                                             (startz       ?startz)
                                             (centroidx   =(avg ?startx ?node3x))
                                             (centroidy   =(avg ?starty ?node3y))
                                             (centroidz   =(avg ?startz ?node3z))
                                             (sidecount     1)
                                             (accuracy     ?polyaccuracy)
                                             (trait         HIDDEN_EDGE)
                                             (classification WALL))))
               )

               (bind ?length (distance ?node2x ?node2y ?node2z ?node3x ?node3y ?node3z))

            ;   if distance is close enough, then test colinear/convex/concave
               (if    (<= ?distance ?*max_edge_distance*)
                then (if    (<= (abs (normalize2 (- ?orientation1 ?orientation2)))
                                ?*max_edge_angle*)
                     then ; colinear edge found and added to polyhedron
                          ; also add 'S' area between start point and new segments
                          (bind ?trianglearea1 (S ?startx ?starty
                                                  ?node1x ?node1y ?node2x ?node2y))
                          (bind ?trianglearea2 (S ?startx ?starty
                                                  ?node2x ?node2y ?node3x ?node3y))
                          (modify ?poly  (end    ?endline2time)
                                         (area  =(+ ?area ?trianglearea1 ?trianglearea2))
                                         (centroidx  =(+ ?centroidx ?node2x ?node3x))
                                         (centroidy  =(+ ?centroidy ?node2y ?node3y))
                                         (centroidz  =(+ ?centroidz ?node2z ?node3z))

                                         (sidecount  =(+ ?sidecount  1)))
                          (modify ?line2 (status USED))

                          (assert (Polyhedron (status        COMPLETE)
                                              (start        ?endline1time)
                                              (end          ?startline2time)
                                              (startx       ?node1x)
                                              (starty       ?node1y)
                                              (startz       ?node1z)
                                              (centroidx   =(avg ?node1x ?node2x))
                                              (centroidy   =(avg ?node1y ?node2y))
                                              (centroidz   =(avg ?node1z ?node2z))
                                              (sidecount     1)
                                              (accuracy    =(max ?accuracy1 ?accuracy2))
                                              (trait         INFERRED_EDGE)
                                              (classification WALL)))
                          (if    (>= ?length ?*min_wall_length*)
                           then (output_polyhedron  ?startline2time ?endline2time
                                                    ?node2x ?node2y 0.0
                                                    ?node3x ?node3y 8.0
                                                    "WALL"
                                                    "(added colinear edge)"))
```

```
else ; test for convex edge to continue building,
     ; otherwise edge is concave and polyhedron is complete.
     ; note this rule currently coded to work only for left transducer

     (if   (< (normalize2 (- ?orientation2 ?orientation1)) 0.0)

      then ; convex edge found, and joined to polyhedron
           ; also add 'S' area between start point and new segments
           (bind ?trianglearea1 (S ?startx ?starty
                                   ?node1x ?node1y ?node2x ?node2y))
           (bind ?trianglearea2 (S ?startx ?starty
                                   ?node2x ?node2y ?node3x ?node3y))
           (modify ?poly  (end           ?endline2time)
                          (classification  OBJECT)
                          (area        =(+ ?area ?trianglearea1
                                               ?trianglearea2))
                          (accuracy    =(max ?accuracy2
                                             ?accuracy3 ?polyaccuracy))
                          (centroidx   =(+ ?centroidx ?node2x ?node3x))
                          (centroidy   =(+ ?centroidy ?node2y ?node3y))
                          (centroidz   =(+ ?centroidz ?node2z ?node3z))
                          (sidecount   =(+ ?sidecount  1)))
           (modify ?line2 (status USED))

           (assert (Polyhedron (status       COMPLETE)
                               (start        ?endline1time)
                               (end          ?startline2time)
                               (startx       ?node1x)
                               (starty       ?node1y)
                               (startz       ?node1z)
                               (centroidx   =(avg ?node1x ?node2x))
                               (centroidy   =(avg ?node1y ?node2y))
                               (centroidz   =(avg ?node1z ?node2z))
                               (sidecount    1)
                            (accuracy    =(max ?accuracy1 ?accuracy2))
                               (trait         INFERRED_EDGE)
                               (classification WALL)))

           (if    (>= ?length ?*min_wall_length*)
            then (output_polyhedron  ?startline2time ?endline2time
                                     ?node2x ?node2y 0.0
                                     ?node3x ?node3y 8.0
                                     "WALL"
                                     "(added convex edge)"))

      else ; concave edge found so don't continue building polyhedron
           (modify ?poly (status COMPLETE)
                         (area        =(abs ?area))
                         (centroidx  =(/ ?centroidx ?sidecount 2))
                         (centroidy  =(/ ?centroidy ?sidecount 2))
                         (centroidz  =(/ ?centroidz ?sidecount 2))
                         (sidecounter1   ?sidecount)
                         (sidecounter2   ?sidecount))

           ; if polyhedron was not a WALL, assert a HIDDEN_EDGE wall
           (if    (not (eq ?classification WALL))
            then (format t "%nPolyhedron OBJECT (%3.1f .. %3.1f) "
                           ?startpolytime ?endpolytime)
                 (format t "has area %3.1f%n" (abs ?area))
          (format auvfile "  (prior object area was %3.1f)" (abs ?area))
                 (assert (Polyhedron (status  COMPLETE)
                               (start        ?startpolytime)
                               (end          ?endpolytime)
                               (startx       ?startx)
                               (starty       ?starty)
                               (startz       ?startz)
                               (centroidx   =(avg ?startx ?node3x))
                               (centroidy   =(avg ?starty ?node3y))
```

```
                                    (centroidz    =(avg ?startz ?node3z))
                                    (sidecount      1)
                                    (accuracy     ?polyaccuracy)
                                    (trait          HIDDEN_EDGE)
                                    (classification WALL))))
              )))
)

;_______________________________________________________________________________

(defrule complete-active-polyhedron-after-file-reading-finished

   (declare (salience 420))   ; polyhedron determination rules take precedence
   ?range-file-closed <- (range-file-closed-flag)

   ?poly <- (Polyhedron (status          ACTIVE)
                        (start          ?startpolytime)
                        (end            ?endpolytime)
                        (startx         ?startx)
                        (starty         ?starty)
                        (startz         ?startz)
                        (accuracy       ?polyaccuracy)
                        (centroidx      ?centroidx)
                        (centroidy      ?centroidy)
                        (centroidz      ?centroidz)
                        (sidecount      ?sidecount)
                        (area           ?area)
                        (classification ?classification))

;   node matches end of polyhedron
   ?node <- (Node (time  ?nodetime)
                  (accuracy ?accuracy)
                  (x ?nodex) (y ?nodey) (z ?nodez))
   (test (= ?endpolytime ?nodetime))

=>
    (modify ?poly (status COMPLETE)
                  (area       =(abs ?area))
                  (centroidx  =(/ ?centroidx ?sidecount 2)) ; 2 points/side
                  (centroidy  =(/ ?centroidy ?sidecount 2))
                  (centroidz  =(/ ?centroidz ?sidecount 2))
                  (sidecounter1   ?sidecount)
                  (sidecounter2   ?sidecount))

   ; if polyhedron was not a WALL, assert a HIDDEN_EDGE wall for it
    (if   (not (eq ?classification WALL))
     then (format t "%nPolyhedron OBJECT (%3.1f .. %3.1f) has area %3.1f%n"
                    ?startpolytime ?endpolytime (abs ?area))
          (format auvfile "  (prior object area was %3.1f)" (abs ?area))
          (assert (Polyhedron (status  COMPLETE)
                              (start         ?startpolytime)
                              (end           ?endpolytime)
                              (startx        ?startx)
                              (starty        ?starty)
                              (startz        ?startz)
                              (centroidx   =(avg ?startx ?nodex))
                              (centroidy   =(avg ?starty ?nodey))
                              (centroidz   =(avg ?startz ?nodez))
                              (sidecount      1)
                              (accuracy      ?polyaccuracy)
                              (trait          HIDDEN_EDGE)
                              (classification WALL))))
)

;_______________________________________________________________________________
;
; Completed polyhedron area calculation rules
;_______________________________________________________________________________
```

```
(defrule oldarea1

; compute-polyhedron-area-contribution-from-regression-edges

   (declare (salience 415))   ; polyhedron determination rules take precedence
   ?poly <- (Polyhedron (status         COMPLETE)
                        (trait          OBJECT_BUILDING_BASED)
                        (classification OBJECT)
                        (start         ?startpolytime)
                        (end           ?endpolytime)
                        (startx        ?startx)
                        (starty        ?starty)
                        (startz        ?startz)
                        (centroidx     ?node3x)
                        (centroidy     ?node3y)
                        (centroidz     ?node3z)
                        (sidecount     ?sidecount)
                        (sidecounter1  ?sidecounter1)
                        (area          ?area))

   (test (> ?sidecounter1 0)) ; prevent infinite recursion

;  get the next line contributing to polyhedron area
   ?edge <- (Edge (start      ?startedgetime)
                  (end        ?endedgetime)
                  (status      USED))
   (test (and (>= ?startedgetime ?startpolytime)
              (<= ?endedgetime   ?endpolytime)))

;   node1 matches start of edge
   ?node1 <- (Node (time  ?node1time)
                   (x ?node1x) (y ?node1y) (z ?node1z))
   (test (= ?startedgetime ?node1time))

;   node2 matches end of edge
   ?node2 <- (Node (time  ?node2time)
                   (x ?node2x) (y ?node2y) (z ?node2z))
   (test (= ?endedgetime ?node2time))

=>

   (bind ?trianglearea (S ?node1x ?node1y ?node2x ?node2y ?node3x ?node3y))

   (modify ?poly (sidecounter1 =(- ?sidecounter1 1))
                 (area         =(+ ?area (abs ?trianglearea))))

   (modify ?edge (status        USED_FOR_AREA))

)
;_________________________________________________________________________________

(defrule  oldarea2

; compute-polyhedron-area-contribution-from-inferred-walls

   (declare (salience 410))   ; polyhedron determination rules take precedence
   ?poly <- (Polyhedron (status         COMPLETE)
                        (trait          OBJECT_BUILDING_BASED)
                        (classification OBJECT)
                        (start         ?startpolytime)
                        (end           ?endpolytime)
                        (startx        ?startx)
                        (starty        ?starty)
                        (startz        ?startz)
                        (centroidx     ?node3x)
                        (centroidy     ?node3y)
```

```
                         (centroidz     ?node3z)
                         (sidecount     ?sidecount)
                         (sidecounter2  ?sidecounter2)
                         (area          ?area))

   (test (> ?sidecounter2 0)) ; prevent infinite recursion

;  get a matching inferred wall or hidden edge polyhedron
   ?poly2 <- (Polyhedron (status      COMPLETE)
                         (trait       INFERRED_EDGE | HIDDEN_EDGE)
                         (classification WALL)
                         (start      ?startpoly2time)
                         (end        ?endpoly2time)
                         (startx     ?node1x)
                         (starty     ?node1y)
                         (startz     ?node1z))
   (test (and (>= ?startpoly2time ?startpolytime)
              (<= ?endpoly2time   ?endpolytime)))

;   node2 matches the end of this inferred/hidden wall
   ?node2 <- (Node (time  ?node2time)
                   (x ?node2x) (y ?node2y) (z ?node2z))
   (test (= ?endpoly2time ?node2time))

=>
   (bind ?trianglearea (S ?node1x ?node1y ?node2x ?node2y ?node3x ?node3y))

   (modify ?poly  (sidecounter2 =(- ?sidecounter2 1))
                  (area         =(+ ?area (abs ?trianglearea))))

   (modify ?poly2 (status        USED_FOR_AREA))

   (if    (eq ?sidecounter2 1) ; last edge triangle has been added
    then  (format t "%nPolyhedron OBJECT (%3.1f .. %3.1f) has area %3.1f%n"
                    ?startpolytime ?endpolytime (+ ?area (abs ?trianglearea))))

)

;________________________________________________________________________________
;
; Object classification rules:  the top level at last!
;________________________________________________________________________________

(defrule classify-pool-objects

   (declare (salience 400))   ; polyhedron determination rules take precedence

   ?poly <- (Polyhedron (status         COMPLETE)
                        (trait          OBJECT_BUILDING_BASED)
                        (classification OBJECT)
                        (start          ?startpolytime)
                        (end            ?endpolytime)
                        (startx         ?startx)
                        (starty         ?starty)
                        (startz         ?startz)
                        (centroidx      ?centroidx)
                        (centroidy      ?centroidy)
                        (centroidz      ?centroidz)
                        (sidecount      ?sidecount)
                        (area           ?area))

;   node matches end of polyhedron
   ?node <- (Node (time     ?nodetime)
                  (accuracy ?accuracy)
                  (x ?endx) (y ?endy) (z ?endz))
   (test (= ?endpolytime ?nodetime))

=>
```

```
; - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

;  Reclassify long skinny objects as walls

   (bind ?length (distance ?startx ?starty ?startz ?endx ?endy ?endz))
   (if    (<= (/ ?area ?length ?length) ?*wall_thinness_ratio*)
    then (bind  ?area 0.0)
         (modify ?poly (classification WALL) (area 0.0))
         (format t "%n*** OBJECT at (%3.1f .. %3.1f) reclassified as a WALL.%n"
                         ?startpolytime ?endpolytime)
   )

; - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

;  Mine classification

   (if    (and (>= ?area 10.0) (<= ?area 100.0)) ; area criteria test
    then (modify ?poly (classification MINE))
         (format   t "%n%nThe polyhedron at times (%3.1f .. %3.1f) "
                      ?startpolytime ?endpolytime)
         (printout t "has classification MINE." crlf crlf)
         (format auvfile
          "%n%s    %5.1f %4.1f %3.1f  %5.1f          time %4.1f"
                  MINE
                  (+ ?centroidx ?*offsetx*)
                  (+ ?centroidy ?*offsety*)
                  (+ ?centroidz ?*offsetz*)
                  (/ ?area (pi) 2 6) ; radius
                  ?endpolytime)
         (format auvfile "%n")

         (format ?*out*
          "%n%s    %5.1f %4.1f %3.1f  %5.1f          time %4.1f"
                  Mine
                  (+ ?centroidx ?*offsetx*)
                  (+ ?centroidy ?*offsety*)
                  (+ ?centroidz ?*offsetz*)
                  (/ ?area (pi) 2 6) ; radius & sonar beamwidth fudge factor
                  ?endpolytime)
         (format ?*out* "%n"))
)

;____________________________________________________________________________
```

LIST OF REFERENCES

[Ref. 1] Brutzman, D.P., Compton, M.A., "AUV Research at the Naval Postgraduate School," *Sea Technology*, v. 32, n.12, pp. 35-40, December 1991.

[Ref. 2] Hartmann, G.K., Truver, S.C., *Weapons that Wait*, Updated Edition, Naval Institute Press, Annapolis, Maryland, 1991.

[Ref. 3] Horne, C.F., "Mine Warfare Is With Us and Will Be With Us," *Proceedings*, U.S. Naval Institute, v. 117/7/1061, p.63, July 1991.

[Ref. 4] Good, Lt. Michael R., *Design and Construction of a Second Generation AUV*, Master's Thesis, Naval Postgraduate School, Monterey, California, 1989.

[Ref. 5] Healey, A.J., McGhee, R.B., Christi, R., Papoulias, F.A., Kwak, S.H., Kanayama, Y., Lee, Y., "Mission Planning, Execution and Data Analysis for the NPS AUV II Autonomous Underwater Vehicle," *Proceedings* of 1st IARP Workshop on Mobile Robots for Subsea Environments, Monterey, California (1991), pp. 177-186.

[Ref. 6] Floyd, C.A., *Design and Implementation of a Collision Avoidance System for the NPS Autonomous Underwater Vehicle (AUVII) Utilizing Ultrasonic Sensors*, Master's Thesis, Naval Postgraduate School, Monterey, California, 1989.

[Ref. 7] Brutzman, D.P., Compton, M.A., Kanayama, Y., "Autonomous Sonar Classification using Expert Systems," draft article, OCEANS 92 conference, Oceanic Engineering Society of the IEEE, Newport, Rhode Island, October,1992.

[Ref. 8] Ong, S.M., *A Mission Planning Expert System with Three-Dimensional Path Optimization for the NPS Model 2 Autonomous Underwater Vehicle*, Master's Thesis, Naval Postgraduate School, Monterey, California, 1990.

[Ref. 9] Boncal, R.J., *A Study of Model Based Maneuvering Controls for Autonomous Underwater Vehicles*, Master's Thesis, Naval Postgraduate School, Monterey, California, 1987.

[Ref. 10] MacPherson, D.L., *A Computer Simulation Study of Mission Planning and Control for the NPS Autonomous Underwater Vehicle*, Master's Thesis, Naval Postgraduate School, Monterey, California, 1988.

[Ref. 11] Jurewicz, T.A., *A Real Time Autonomous Underwater Vehicle Dynamic Simulator*, Master's Thesis, Naval Postgraduate School, Monterey, California, 1990.

[Ref. 12] Clotier, M., *Guidance and Control System for an Autonomous Vehicle*, Master's Thesis, Naval Postgraduate School, Monterey, California, 1990.

[Ref. 13] Magrino, C., *Three Dimensional Guidance for the NPS Autonomous Underwater Vehicle*, Master's Thesis, Naval Postgraduate School, Monterey, California, 1991.

[Ref. 14] Brutzman, D.P., *NPS AUV Integrated Simulator*, Master's Thesis, Naval Postgraduate School, Monterey, California, 1992.

[Ref. 15] Pappas, G., Shotts, W., O'Brien, M, Wyman, W., "The DARPA/Navy Unmanned Undersea Vehicle Program," *Unmanned Systems*, v.9,n.2, pp. 24-30, Spring 1991.

[Ref. 16] *Autonomous Minehunting Technology*, AMT Program Review Notes 1991, Defense Advanced Research Projects Agency and Charles Stark Draper Laboratories, Cambridge, Massachusetts, December 11, 1991.

[Ref. 17] Bernstein, J., *Micromechanical Hydrophone*, AMT Program Review Notes 1991, Charles Stark Draper Laboratories, Cambridge, Massachusetts.

[Ref. 18] RBCV-TR-88-23, *University of Toronto, Robotic Exploration as Graph Construction*, by G. Dudek, M. Jenkin, E. Milios and D. Wilkes, November 1988.

[Ref. 19] Dossey, J., and others, *Discrete Mathematics*, pp. 98-102, Scott, Foresman and Company, 1987.

[Ref. 20] Department of the Navy, *NWP55-8-SAR*/NAVAIR A1-SARBA-TAC-000, rev. B, February 1990.

[Ref. 21] Rowe, N., *Artificial Intelligence Through Prolog*, pp. 191-214, Prentice Hall, Englewood Cliffs, New Jersey, 1988.

[Ref. 22] Hart, P.E., Nilsson, N.J., Raphael, B., "A Formal Basis for the Heuristic Determination of Minimum Cost Paths," *IEEE Transactions of Systems, Man, and Cybernetics*, v. SSC-4, n. 2, pp. 100-107, July 1968.

[Ref. 23] Naval Postgraduate School Graphics Programming Project, *Submarine Launched AUV Concept Project and Modeling the Sonar Environment*, by Compton, M.A. and Cadwallader, N., September 1991.

[Ref. 24] Chappell, S.G., "A Simple World Model for an Autonomous Vehicle," *Proceedings* of the 6th International Symposium on Unmanned Untethered Submersible Technology, Marine Systems Engineering Laboratory, University of New Hampshire, Durham, New Hampshire, p. 512.

[Ref. 25] *The TIMES Atlas and Encyclopedia of the Sea*, Time Books Limited, 1989.

[Ref. 26] Zehner, W.J., Loggins, C.D., "Selection Criteria for UUV Sonar Systems," *Proceedings* of the 6th International Symposium on Unmanned Untethered Submersible Technology, Marine Systems Engineering Laboratory, University of New Hampshire, Durham, New Hampshire, pp. 350-358.

[Ref. 27] McKeon, J., *Incorporation of GPS/INS into Small Autonomous Underwater Vehicle Navigation,* Master's Thesis, Naval Postgraduate School, Monterey, California, 1992.

[Ref. 28] Compton, M., "Modeling the Sonar Environment," unpublished paper, Naval Postgraduate School, Monterey, California, September 1991.

[Ref. 29] Herbert, M., Kanade, T., Kweon, I., "3-D Vision Techniques for Autonomous Vehicles," NSF Range Image Understanding Workshop, pp. 273-337, 1988.

[Ref. 30] Besl, P.J., Jain, R.C., "Three-Dimensional Object Recognition," Computing Surveys, v. 17, n. 1, pp. 77-145, March 1985.

[Ref. 31] Iyengar, S., Elfes, A., "Autonomous Underwater Robots: Perception, Mapping and Navigation", IEEE Computer Society Press, v. 1, Los Alamitos, California, 1991.

[Ref. 32] Luo, R., Kay, M., "Multisensor Integration and Fusion in Intelligent Systems," IEEE Transactions on Systems, Man and Cybernetics, v. 19, n. 5, pp. 901-931, September/October 1989.

[Ref. 33] Moravec, H., "The Stanford Cart and the CMU Rover," *Proceedings* of the IEEE, v. 71, pp. 872-884, July 1983.

[Ref. 34] Stewart, K., "Three-Dimensional Modeling of Seafloor Backscatter from Sidescan Sonar for Autonomous Classification and Navigation," *Proceedings* of the 6th International Symposium on Unmanned Untethered Submersible Technology, University of New Hampshire, Durham, New Hampshire, pp. 372-392, June 1989.

[Ref. 35] Blidberg, D.R., Chappell, S., Jalbert, J., Turner, R., Sedor, G., Eaton, P., "The EAVE AUV Program at the Marine Systems Engineering Laboratory," *Proceedings* of 1st IARP Workshop on Mobile Robots for Subsea Environments, Monterey, California, pp. 33-42, October 1990.

[Ref. 36] Floyd, C., Kanayama, Y., Magrino, C., "Underwater Obstacle Recognition using a Low-Resolution Sonar," *Proceedings* of the Seventh International Symposium on Unmanned Untethered Submersible Technology, University of New Hampshire, Durham, New Hampshire, pp. 309-327, September 1991.

[Ref. 37] Kanayama, Y., Noguchi, T., "Spatial Learning by an Autonomous Mobile Robot with Ultrasonic Sensors," University of California Santa Barbara Department of Computer Science Technical Report TRCS86-06, February 1989.

[Ref. 38] Kanayama, Y., Noguchi, T., Hartman, B., "Sonar Data Interpretation for Autonomous Mobile Robots," unpublished paper, Naval Postgraduate School, Monterey, California, 1990.

[Ref. 39] Jackson, P., *Introduction to Expert Systems*, Addison-Wesley Publishing Co. Inc., Workingham, England, 1991.

[Ref. 40] Sacerdoti, E., "Managing Expert System Development," AI Expert, v. 6, n. 5, pp. 26-33, May 1991.

[Ref. 41] Brutzman, D., Floyd, C. Whalen, R., "Naval Postgraduate School Autonomous Underwater Vehicle," Video *Proceedings* of the IEEE International Conference on Robotics and Automation 92, Nice, France, May 1992.

[Ref. 42] Brutzman, D., "Integrated Simulation for Rapid Development of Autonomous Underwater Vehicles," *Proceedings* of the IEEE Oceanic Engineering Society Conference AUV 92, Washington DC, June 1992.

[Ref. 43] NASA Software Technology Branch, *Clips Reference Manual*, Lyndon B. Johnson Space Center, Houston, Texas, 1991.

[Ref. 44] Giarratano, J., *CLIPS User's Guide*, NASA, Lyndon B. Johnson Space Center, January 1991.

[Ref. 45] Brooks, T., "The Art of Production Systems," AI Expert, v. 7, n. 1, pp. 30-35, January 1992.

[Ref. 46] Corkill, D., "Blackboard Systems," AI Expert, v. 6, n. 9, pp. 40-47, September 1991.

INITIAL DISTRIBUTION LIST

1.	Defense Technical Information Center, Cameron Station, Alexaneria, VA 22304-6145	2
2.	Dudley Knox Library, Code 52, Naval Postgraduate School, Monterey, CA 93943-5002	2
3.	Commander, Mine Warfare Command, Charleston, SC 29408	1
4.	Chief of Naval Research, 800 North Quincy Street, Arlington, VA 22217-5000	1
5.	Commander, Naval Sea Systems Command, ATTN: CAPT William Shotts, PMO-403, Washington, DC 20362-5101	1
6.	RADM George R. Sterner USN, Program Executive Officer, Submarine Combat and Weapons Systems, Department of the Navy, Washington, DC 20362-5101	1
7.	Commander, Submarine Development Squadron TWELVE, Naval Submarine Base, Groton, CT 06340	1
8.	Commanding Officer, Patrol Squadron FORTY, FPO San Francisco 96601-5916	1
9.	Commanding Officer, Naval Underwater Systems Center, Newport, RI 02841-5047	1
10.	Commanding Officer, Naval Coastal Systems Center, Panama City, FL 32407-5000	1

11. Commander 1
Naval Surface Weapons Center
Dahlgren, VA 22448-5000

12. CAPT Alan R. Beam USN 1
DARPA UWO - PRC Inc.
1555 Wilson Boulevard
Suite 600
Arlington, VA 22209

13. Dr. Richard Guertin 1
OP-09BC
Pentagon 4D386
Washington, DC 20301-5000

14. MAJ David Neyland USAF 1
DARPA ASTO
3701 North Fairfax Drive
Arlington, VA 22203

15. Commanding Officer 1
David Taylor Research Center
Bethesda, MD 20084-5000

16. Commander 1
Naval Oceans Systems Center
San Diego, CA 92152-5000

17. Mr. Randy Brill 1
Naval Oceans Systems Center
PO Box 997
Kailua, HI 96734-0996

18. Director 1
Navy Center for Applied Research in Artificial Intelligence
Naval Research Laboratory
Washington, DC 20375-5000

19. Mr. Patrick Hale 1
DARPA UUV Program Manager
C.S. Draper Laboratories
555 Technology Square
Cambridge, MA 02139

20. Dr. D. Richard Blidberg 1
Marine Systems Engineering Laboratory
Marine Program Building
University of New Hampshire
Durham, NH 03824-3525

21. Dr. James G. Bellingham 1
Sea Grant College Program
Massachusetts Institute of Technolog
292 Main Street
Cambridge, MA 02139

22. Dr. Dana R. Yoerger 1
Deep Submergence Laboratory
Department of Applied Ocean Physics and Engineering
Woods Hole Oceanographic Institute
Woods Hole, MA 02543

23. Dr. Robert B. McGhee 1
Code CS/Mz
Chairman, Computer Science Department
Naval Postgraduate School
Monterey, CA 93943-5000

24. Dr. Anthony J. Healey 1
Code ME/Hy
Chairman, Mechanical Engineering Department
Naval Postgraduate School
Monterey, CA 93943-5000

25. Dr. Man-Tak Shing 1
Code CS/Sh
Computer Science Department
Naval Postgraduate School
Monterey, CA 93943-5000

26. Dr. Yutaka Kanayama 1
Code CS/Ka
Computer Science Department
Naval Postgraduate School
Monterey, CA 93943-5000

27. Dr. Neil C. Rowe 1
Code CS/Rp
Computer Science Department
Naval Postgraduate School
Monterey, CA 93943-5000

28. Dr. Se-Hung Kwak 1
Code CS/Kw
Computer Science Department
Naval Postgraduate School
Monterey, CA 93943-5000

29.	LCDR Donald P. Brutzman Code OR/Br Operations Research Department Naval Postgraduate School Monterey, CA 93943-5000	1
30.	LCDR Mark A. Compton 630 W. Garland Terrace Sunnyvale, CA 94086	1